UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

PhD Thesis

Sublinear Algorithms, Online Packing, Hashing and Clustering

Lorenzo Beretta

Supervisor
Mikkel Thorup
Co-Supervisor
Mikkel Abrahamsen

This thesis has been submitted to the PhD School of The Faculty of Science, University of Copenhagen.

Submission date: January 17, 2024.

Funding

This research is supported by Thorup's Investigator Grant from the Villum Foundation under Grant No. 16582, Basic Algorithms Research Copenhagen (BARC), Denmark.

Lorenzo Beretta has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 801199.

Abstract

In this thesis we study the theoretical properties of several algorithmic problems. The first two problems aim at completing a certain task in sublinear time, namely using a number of operation that is sublinear in the input size. A short description of these tasks follows.

- *Earth Mover's Distance:* We are given two distribution μ, ν over a generic metric space $(\mathcal{M}, d)$. A transport plan between μ and ν is a distribution ξ over $\mathcal{M}^2$ that has μ and ν as its marginals. We define the cost of a transport plan ξ as $E_{(x,y)\sim\xi}[d(x,y)]$. Our task is to compute the minimum cost of transport plans between μ and ν.
- *Sum estimation via weighted sampling:* We are given a set U of items and each item $u \in U$ has a weight $w(u)$. We can sample elements from U with probability proportional to their weight. Our task is to estimate the combined weight $\sum_{u\in U} w(u)$ using as few samples as possible.

Then, we study two problems concerned with online packing of polygons. In both cases, we give upper and lower bounds on the competitive ratio of these problems, described below.

- *Online packing of rectangles:* We are given a sequence of axis-parallel rectangles online and we have to irrevocably place them on the plane so as to minimize the area or perimeter of their axis-parallel bounding box.
- *Translational packing of convex polygons:* We are given a sequence of convex polygons online and we have to irrevocably place them in a given container on the plane without rotating them. The goal is to use as little container space as possible. We prove a surprising superconstant lower bound on the competitive ratio for several well-studied container types.

Besides packing, we study hash functions.

- *Locally uniform hashing:* We design Tornado tabulation, a new tabulation-based hash function that aims at filling the gap between (unrealistic) fully-random hashing often used in the analysis of algorithms and practical hash functions with weak theoretical guarantees. In theory, Tornado has strong distributional properties, ensuring that its keys enjoy a certain local full randomness. Moreover, Tornado is practical and it is implementable in a few lines of C code.

Finally, we include a result on clustering.

- *Local-search seeding for k-means:* The most popular heuristic for k-means clustering, Lloyd's algorithm, inherits its theoretical guarantees from the seeding strategy employed to initialize cluster centers. We design a local-search algorithm to initialize cluster centers that improves the approximation ratios of previous seeding strategies. We claim that our algorithm is practical and include experiments to support its effectiveness.

Resumé

I denne afhandling undersøger vi de teoretiske egenskaber ved adskillige algoritmiske problemer. De første to problemer sigter mod at fuldføre en bestemt opgave på sublineær tid, nemlig ved hjælp af et antal operationer, der er sublineære i inputstørrelsen. En kort beskrivelse af disse opgaver følger.

- *Earth Mover's Distance:* Vi har to fordelinger μ, ν over et generisk metrisk rum $(\mathcal{M}, d)$. En transportplan mellem μ og ν er en fordeling ξ over $\mathcal{M}^2$, der har μ og ν som dens margener. Vi definerer omkostningen ved en transportplan ξ som $E_{(x,y)\sim\xi}[d(x,y)]$. Vores opgave er at beregne den minimale omkostning ved transportplaner mellem μ og ν.
- *Sum estimation via weighted sampling:* Vi har en mængde U af elementer, og hvert element $u \in U$ har en vægt $w(u)$. Vi kan udtrække elementer fra U med sandsynlighed proportionel med deres vægt. Vores opgave er at estimere den samlede vægt $\sum_{u\in U} w(u)$ ved at bruge så få prøver som muligt.

Derefter undersøger vi to problemer vedrørende online-pakning af polygoner. I begge tilfælde giver vi øvre og nedre grænser for den konkurrencebaserede ratio af disse problemer, som beskrevet nedenfor.

- *Online packing of rectangles:* Vi får en sekvens af aksialt parallelle rektangler online, og vi skal uigenkaldeligt placere dem på planet for at minimere området eller omkredsen af deres aksialt parallelle begrænsningsboks.
- *Translational packing of convex polygons:* Vi får en sekvens af konvekse polygoner online, og vi skal uigenkaldeligt placere dem i en given beholder på planet uden at rotere dem. Målet er at bruge så lidt beholderplads som muligt. Vi beviser en overraskende superkonstant nedre grænse for konkurrenceforholdet for flere velundersøgte beholder typer.

Udover pakning studerer vi hashfunktioner.

- *Locally uniform hashing:* Vi designer Tornado-tabulering, en ny tabuleringsbaseret hashfunktion, der sigter mod at udfylde kløften mellem (urealistisk) fuldstændig tilfældig hashing, der ofte bruges i analyse af algoritmer, og praktiske hashfunktioner med svage teoretiske garantier. Teoretisk set har Tornado stærke distributionsmæssige egenskaber, der sikrer, at dens nøgler nyder en vis lokal fuldstændig tilfældighed. Desuden er Tornado praktisk og kan implementeres på få linjer C-kode.

Endelig inkluderer vi et resultat om clustering.

- *Local-search seeding for k-means:* Den mest populære heuristik for k-means clustering, Lloyds algoritme, arver sine teoretiske garantier fra den såede strategi, der bruges til at initialisere klyngecentre. Vi designer en lokal søgealgoritme til at initialisere klyngecentre, der forbedrer approksimationsforholdene for tidligere såningsstrategier. Vi hævder, at vores algoritme er praktisk, og inkluderer eksperimenter for at understøtte dens effektivitet.

Preface

The *General rules and guidelines for the PhD programme* at the Faculty of Science, University of Copenhagen allows for a PhD dissertation to be written *"as a synopsis with manuscripts of papers or already published papers attached"*. The present dissertation has this form.

Throughout my PhD, I have written 5 published papers and 1 unpublished manuscript and I include all of them. We can subdivide these 6 papers into three thematic blocks. The first two blocks constitutes the core theme of my PhD thesis: the first block of two papers is concerned with sublinear-time algorithms, while the second block of two papers is concerned with online packing of polygons on the plane. I included a third block of two papers that are concerned with hashing and geometric clustering respectively. This last block can be interpreted as a "miscellana" that showcases the breadth of my PhD journey and my exploration of other beautiful areas within theoretical computer science.

Included in the thesis

Sublinear Algorithms. In modern data analysis and machine learning, datasets are huge. When the size of the dataset is so large that it does not fit in main memory, classical algorithms designed for the RAM model of computation are no longer practical. As a response, theoreticians came up with new models of computation capturing technologically feasible approaches like sampling a small part of the input (property testing), reading through the data only once (streaming) or distributed computation (MPC). If an algorithm is not allowed to read (or store) the whole input then we say that it is a *sublinear algorithm.* In this block we design sublinear algorithms for two problems.

Earth Mover's Distance: We are given two distribution μ, ν over a generic metric space $(\mathcal{M}, d)$. A transport plan between μ and ν is a distribution ξ over $\mathcal{M}^2$ that has μ and ν as its marginals. We define the cost of a transport plan ξ as $E_{(x,y)\sim\xi}[d(x,y)]$ and say that the earth mover's distance (EMD) between μ and nu is the minimum cost of transport plans between μ and ν. We design an algorithm that computes $\text{EMD}(\mu, \nu)$ up to a ε-additive approximation in time $O(n^{2-\delta(\varepsilon)})$, where n is (an upper bound to) the support size of μ and ν and $\delta(\varepsilon) > 0$ as long as $\varepsilon > 0$. Notice that, since the input representation includes a $n \times n$ distance matrix, our algorithm takes sublinear time in the input size.

Sum estimation via weighted sampling: Given a large set U where each item $u \in U$ has weight $w(u)$, we want to estimate the total weight $W = \sum_{u \in U} w(u)$ to within factor of $1 \pm \varepsilon$ with some constant probability $> 1/2$. Since $n = |U|$ is large, we want to do this without looking at the entire set U. In the traditional setting in which we are allowed to sample elements from U uniformly, sampling $\Omega(n)$ items is necessary to provide any non-trivial guarantee on the estimate. Therefore, we investigate this problem in different settings: in the *proportional* setting we can sample items with probabilities proportional to their weights, and in the *hybrid* setting we can sample both proportionally and uniformly. These settings have applications such as counting the number of edges in large graphs.

Online Packing of Polygons. Packing problems are omnipresent in our daily lives and likewise appear in many large-scale industries. For instance, two-dimensional versions of packing arise when a given set of pieces has to be cut out from a large piece of material such that waste is minimized. In this block, we consider the problem of packing a stream of polygons on the plane *online*, namely placing them irrevocably on the plane as they come. The objective of our algorithm is to use as little space as possible. The metric used to evaluate our algorithms' performance is the competitive ratio, that is the ratio between the cost of the solution obtain online by the algorithm and the optimal offline solution. In particular, we study the two following problems.

Online packing of rectangles: Given a stream of online axis-parallel rectangles, our goal is to place them on the plane without overlaps so as to minimize the area (resp. perimeter) of their axis-parallel bounding box. We study variants of this problem where we are allowed to rotate the rectangles and where we are not and we give tight bounds (up to constant factors) on the competitive ratios for all these problems, both in terms of n and $\textsc{Opt}$.

Online packing of convex polygons, without rotation: Given a stream of convex polygons, we have to place them in a container on the plane (e.g., an 1 strip) online and the goal is to minimize the container space used. We show superconstant lower bounds for several container types. Our lower bounds are proved by reducing our packing problems to a purely combinatorial problem, *online sorting*, that essentially asks to sort a stream of real number online. Interestingly, the same packing problems admit offline constant-approximation algorithms and a key subroutine of these algorithms is that they "sort" pieces according to their natural orientation. Thus, we show that this sorting is, in some sense, necessary to achieve $O(1)$-approximation, and since sorting online is hard then packing is also hard.

Miscellanea: hashing and geometric clustering. In this third block, we present two projects.

Locally uniform hashing: Hashing is a common technique used in data processing, with a strong impact on resources spent on computation. Hashing also affects the applicability of theoretical results that often assume access to (unrealistic) uniform/fully-random hash functions. In this paper, we are concerned with designing hash functions that are practical and come with strong theoretical guarantees on their performance. To this end, we present tornado tabulation hashing, which is simple, fast, and exhibits a certain full, local randomness property that provably makes diverse algorithms perform almost as if (abstract) fully-random hashing was used. For example, this includes classic linear probing, the widely used HyperLogLog algorithm for counting distinct elements [FFGM07], and the one-permutation hashing for large-scale machine learning [LOZ12].

Local-search seeding for k-means: Euclidean k-means is arguably the most popular formulation of center-based clustering. The k-means++ algorithm [AV07] is often the practitioners' choice algorithm for initializing cluster centers before running Lloyd's algorithms [Llo57], and is known to give an $O(\log k)$-approximation algorithm (in expectation) for Euclidean k-means. To obtain higher quality solutions, k-means++ was augmented with $O(k \log \log k)$ local search steps, which yields a c-approximation, where c is a large absolute constant [LS19a]. We generalize and extend the local search algorithm from [LS19a] by considering larger and more sophisticated local search neighborhoods hence allowing to swap multiple centers at the same time. Our algorithm achieves a $9 + \varepsilon$ approximation ratio, which is the best possible for local search. Importantly, we show that our approach yields substantial practical improvements, we show significant quality improvements over [LS19a] on several datasets.

Papers

Approximate Earth Mover's Distance in Truly-Subquadratic Time
Lorenzo Beretta and Aviad Rubinstein
Submitted to STOC 2024 [BR23]

Better Sum Estimation via Weighted Sampling
Lorenzo Beretta and Jakub Tetek
SODA 2022 (Best Student Paper Award) [BT22]

Online Packing to Minimize Area or Perimeter
Mikkel Abrahamsen and Lorenzo Beretta
SoCG 2021 [AB21]

Online Sorting and Translational Packing of Convex Polygons
Anders Aamand, Mikkel Abrahamsen, Lorenzo Beretta, and Linda Kleist
SODA 2023 [AABK23]

Locally Uniform Hashing
Ioana O. Bercea, Lorenzo Beretta, Jonas Klausen, Jakob Bæk Tejs Houen, and Mikkel Thorup
FOCS 2023 [BBK$^+$23]

Multi-Swap k-Means++
Lorenzo Beretta, Vincent Cohen-Addad, Silvio Lattanzi, Nikos Parotsidis
NeurIPS 2023 [BCALP23]

Acknowledgements

First, I would like to express my gratitude to my great supervisors: Mikkel Thorup and Mikkel Abrahamsen. Mikkel Thorup has a great enthusiasm for theoretical computer science and has never missed a chance to share it with me. He introduced me to the field and taught me the importance of being ambitious and always shooting for the stars, so I can at least hit the moon. I would like to thank him particularly for always leaving me free to investigate whatever I wanted, as long as it was *good research*. Finally, thanks Mikkel for fostering the amazing environment at BARC, without your efforts it would not be the same.

Mikkel Abrahamsen is the first person that I worked with at BARC, and I have been extremely lucky to have him as my first co-author and supervisor. He always encouraged me and believed in me when I had just come to Denmark, and I would have not believed in myself. I am extremely grateful for the very many times he spent his time talking to me and teaching me about how to be a researcher. Last but not least, thanks Mikkel for the relationship we built and for always checking on me when I am at BARC.

I would like to thank Aviad, who hosted me at Stanford for six months. The time spent at Stanford has been very happy for me, and I owe this largely to Aviad. Thanks Aviad for writing on your door “please interrupt” and live up to it; thanks for telling me about your kids and asking about my life; thanks for showing me how *you* do research; and finally thanks for inviting me over for Shabbat.

I would like to thank all the BARCies, without the incredibly social environment that you all contribute to create my PhD would not have been nearly as fun. Thanks for all the runs, retreats, celebrations, and everyday lunches. I hope I will find a workplace like BARC in the future.

When I think about these past three years, I am proud of the progress I made as a researcher, and most of this progress I owe to my coauthors, who taught me how to do research. So thanks to Mikkel Abrahamsen, Jakub Tetek, Anders Aamand, Linda Kleist, Ioana O. Bercea, Jonas Klausen, Jakob Bæk Tejs Houen, Mikkel Thorup, Vincent Cohen-Addad, Silvio Lattanzi, Nikos Parotsidis, and Aviad Rubinstein. A special thanks goes to Mikkel, Mikkel, Linda and Aviad, who spent the most energy making me a better scientist.

Finally, I would like to thank my family and all the friends that have been close to me during these three years. In particular, a big thanks to all the people that have been incredibly close even if we were 1,000 km apart.

Contents

Chapter 1

Introduction

In accordance with the guidelines of the PhD School of the University of Copenhagen, this thesis is presented as a synopsis of the papers produced over the course of the PhD program. The main focus is on two topics: sublinear algorithms and online packing of polygons. To demonstrate the breadth of my PhD work, I also include results related to hashing and clustering.

Organisation. The thesis has been divided into six chapters, each one presenting a different paper. The first two are related to sublinear algorithms, the following two focus on online packing of polygons, and the last two are on hashing and clustering respectively.

More specifically, the first chapter develops a sublinear-time algorithm for Earth Mover's Distance. The second chapter develops a sample-optimal estimator for the sum of weights problem. The third chapter discusses online algorithms for packing a stream of axis-parallel rectangles. The fourth chapter discusses algorithms and lower bounds for online packing of general convex polygons. The fifth chapter develops a new hash function, Tornado tabulation hashing. Finally, the sixth chapter covers a new seeding strategy for the popular clustering objective k-means.

The appendix contains the full versions of these 6 papers, in the same order. Each chapter introduces the related problems, provides a survey of the relevant literature, discusses the results of the corresponding paper, and eventually considers future directions and open problems. As each chapter is a synopsis of an actual paper, we refer the reader to the full version of the paper in appendix for proofs as well as any missing detail. We stress that all but one of the presented papers have already been peer reviewed and published.

Chapter 2

Approximate Earth Mover's Distance in Truly-Subquadratic Time

In this chapter, we present the paper "Approximate Earth Mover's Distance in Truly-Subquadratic Time", which is currently under submission at STOC 2024 [BR23]. First, we give a quick introduction to the problem, then we survey some related work and describe our technical contributions. In the end, we point out some interesting future directions.

2.1 Introduction

Given a generic metric space $(\mathcal{M}, d)$ and two probability distributions μ and ν on $\mathcal{M}$ we define the Earth Mover's Distance (EMD) between μ and μ as

$$\mathrm{EMD}(\mu,\nu) = \min\left\{\mathrm{E}_{(x,y)\sim\zeta}[d(x,y)] \;\middle|\; \zeta \text{ is a coupling of } \mu \text{ and } \nu\right\}. \tag{2.1}$$

Here, a distribution ζ over $\mathcal{M}^2$ is a coupling of μ and ν if $\mu(x) = \int_{\mathcal{M}} \zeta(x,y)\,dy$ and $\nu(y) = \int_{\mathcal{M}} \zeta(x,y)\,dx$.

EMD, often called Optimal Transport, is arguably the most natural distance measure for distributions over a metric space and has received significant attention in the machine learning community [V^{+}09, San15, PC19].

If instead of considering EMD over arbitrary distributions we restrict to uniform distributions over two sets of n points, computing EMD becomes equivalent to computing the cost of a minimum-weight perfect matching, a problem with a seventy-year history [Kuh55].

The main result of [BR23] is an algorithm that returns an ε-additive approximation of $\mathrm{EMD}(\mu,\nu)$ in time $O(n^{2-\delta(\varepsilon)})$, where n is (an upper bound to) the support size of μ and ν. Notice that this is the first algorithm to compute EMD in *sublinear time*, indeed for a generic metric the input representation includes an $n \times n$ distance matrix.

2.2 Related Work

Fast computation of Optimal Transport (OT)[1] is a first-class citizen in the agenda of the machine learning community, as it is witnessed by the yearly workshop on OT held at the past few editions of NeurIPS. Some crucial applications of OT emerged in computer vision [RTG00, ACB17, SDGP$^+$15] and natural language processing [KSKW15, YCC$^+$19], where the data is embedded into a vector space as a preprocessing step. For a comprehensive overview of OT applications in modern machine learning we refer to [PC19].

Solving EMD exactly. A recent breakthrough showed an algorithm to solve min-cost flow in near linear time [CKL$^+$22]. Thus, by simply casting EMD as a min-cost flow problem we can solve it exactly in $n^{2+o(1)}$ time. Conversely, any algorithm inspecting less than the whole $n \times n$ matrix cannot solve EMD exactly. Interestingly, a similar fine-grained lower bound holds for $(\log n)$-dimensional Euclidean space, where the input has just size $n \log n$. In fact, Rohatgi proved that no exact algorithm can run in time $O(n^{2-\varepsilon})$, assuming the *strong exponential time hypothesis* [Roh19].

Solving EMD up to multiplicative approximation. In Euclidean space, solving EMD exactly requires time quadratic in the input size, thus an extremely natural question is whether we can provide a multiplicative approximation[2]. A long line of research studied this and related questions [Cha02, Ind03, CJLW22, ACRX22, AS14, AZ23, AIK08, ADBIW09, ANOY14].

In summary, EMD in Euclidean space exhibits a typical dichotomy showing up in geometric optimization problems: in low dimension we have near-linear time approximation schemes, whereas in high dimension the best trade-off between approximation and running time is given by Locality Sensitive Hashing, which achieves c-approximation in time $n^{1+\Theta(1/c)}$ [HPIS13, AZ23].

General metrics are even harder. Indeed, [BCIS05] shows that $o(n^2)$ queries to the distance matrix cannot guarantee any constant approximation. Thus, a next natural question to investigate is whether an additive approximation[3] is achievable in $o(n^2)$ time.

Solving EMD up to additive approximation. For general metrics, additive approximation is natural because, as we have seen, both exact and multiplicative approximation are unachievable in $o(n^2)$ time.

In addition, optimization and machine learning communities have studied additive approximation to EMD for a long time [ANWR17, BJKS18, DGK18, LXH23, Cut13, LNN$^+$21, PLH$^+$20]. This line of research focused on a regularized version of EMD: Sinkhorn distance. On one hand, Sinkhorn has proven very successful in practice, on the other hand optimizing Sinkhorn distance yields an additive approximation to the original EMD objective. Typically, the algorithms from this line of research run in $O_\varepsilon(n^2)$ time and return a ε-additive approximation. Contrarily to the exact min-cost-flow-based solution, these algorithms are practical.

Sublinear algorithms. As mentioned already, the truly-subquadratic complexity of our algorithm is indeed sublinear for general metrics, since the input representation includes a $n \times n$ distance matrix. In [BR23], we initiate the study of EMD in a sublinear model of computation where we have random access to the distance matrix.

[1] Earth Mover's Distance and Optimal Transport are just two names for the same object. The machine learning community prefers OT, whereas the theory community favors EMD. In this thesis we will mainly use EMD.

[2] A C-multiplicative approximation to EMD is a value in $[\mathrm{EMD}(\mu, \nu), C \cdot \mathrm{EMD}(\mu, \nu)]$.

[3] A C-additive approximation to EMD is a value in $[\mathrm{EMD}(\mu, \nu), C + \mathrm{EMD}(\mu, \nu)]$.

Our access model is justified by scenarios where computing a single distance is expensive. For instance, consider the problem of computing the minimum cost of a perfect matching (essentially EMD) with respect to a shortest-path ground metric. If the underlying graph is large, then minimizing the number of distance computation is crucial. In [ALT21], they studied exactly this problem and experimented with heuristics to estimate the minimum-cost of a perfect matching without computing all-pair distances.

In [BR23], we develop the first algorithm that runs a sublinear number of distance computations and returns a provably accurate additive approximation.

2.3 Our Contribution

The main contribution of [BR23] is the following theorem.

Theorem (Theorem 1 from [BR23])**.** *Suppose we have sample access to two distributions μ, ν over metric space $(\mathcal{M}, d)$ satisfying $d(\cdot,\cdot) \in [0,1]$ and query access to d. Suppose further that μ, ν have support size at most n.*

For each constant $\varepsilon > 0$ there exists a constant $f(\varepsilon) > 0$ and an algorithm running in time $O(n^{2-f(\varepsilon)})$ that outputs $\widehat{\mathrm{EMD}}$ such that

$$\widehat{\mathrm{EMD}} \in [\mathrm{EMD}(\mu,\nu) \pm \varepsilon].$$

Moreover, such algorithm takes $\tilde{O}(n)$ samples from μ and ν.

In the rest of this section, we give a sketch of the techniques employed to prove the theorem above. Our proof develops in three steps. First, we reduce EMD to a conceptually simpler problem that we dub min-weight quasi-perfect matching. Second, we design a non-sublinear primal-dual schema that solves the latter. Third, we give a sublinear-time implementation of this schema.

Reduction to min-weight perfect matching. If we take $O_\varepsilon(n)$ samples from both μ and ν and compute the EMD between the empirical distributions of the samples, this yields a ε-additive approximation to $\mathrm{EMD}(\mu,\nu)$. Moreover, since the empirical distributions are actually uniform distributions over a discrete set, this is equivalent to computing he min-weight perfect (bipartite) matching between two sets of points in a metric space. From now on, we will thus focus on the latter problem and aim to approximate the minimum cost of a perfect matching between two sets $A, B \subseteq \mathcal{M}$ of size n. For convenience, we define the cost of the matching M as $\mathrm{cost}(M) = \frac{1}{n}\sum_{e\in M} \mathrm{cost}(e)$.

Min-weight perfect matching with outliers. Since we aim for additive approximation, and we assume $d(\cdot,\cdot) \in [0,1]$, we can afford to leave a small fraction of vertices unmatched. Indeed, if we have a matching M of size $\geq (1-\varepsilon)n$, then extending it with εn arbitrary edges to make it perfect will increase its cost by at most an additive term ε.

This observation about the robustness of the additive approximation version of this problem comes handy while designing our algorithm. Intuitively, whenever a given vertex is hard to handle we can just label it as an outlier and forget about it, as long as we ensure that the total number of outliers is a small fraction of all vertices. In what follows we compute (the cost of) a matching M that is almost perfect (namely, $|M| > (1-\varepsilon)n$) and which cost is not much larger than the minimum cost of any perfect matching.

A non-sublinear template algorithm. First, we recall a classic linear program for min-weight perfect matching. Here we consider a complete bipartite graph on vertices $A \cup B$ and an *integer*[4] cost function $d(\cdot,\cdot) \in [1, C]$. We can interpret the following LP so that $x_{u,v} = 1$ iff u and v are matched, whereas primal constraints require every vertex to be matched.

Primal		**Dual**	
Minimize	$\sum_{u\in A, v\in A} x_{u,v} \cdot c(u,v)$	Maximize	$\sum_{u\in V} \varphi_u$
subject to	$\sum_{v\in B} x_{u,v} \geq 1 \quad \forall u \in A$	subject to	$\varphi_u + \varphi_v \leq c(u,v) \quad \forall u \in A, v \in B$
	$\sum_{u\in A} x_{u,v} \geq 1 \quad \forall v \in B$		$\varphi_u \geq 0. \quad \forall u \in A \cup B,$
	$x_{u,v} \geq 0 \quad \forall u \in A,\ v \in B.$		

Our template algorithm maintains a pair (M, φ), where M is a partial matching and $\{\varphi(v)\}_{v\in A\cup B}$ is a potential associated with each vertex. We maintain the invariant that $\varphi(u)+\varphi(v) \leq c(u,v)+1$ for each $u \in A, v \in B$ and $\varphi(u) + \varphi(v) = c(u,v)$ for each $(u,v) \in M$. Thus, the potential φ is a quasi-feasible dual variable and the the pair (M,φ) satisfies the complementary slackness conditions. If M is a quasi-perfect matching (namely, $|M| \geq (1-\varepsilon)n$) then we also have a quasi-feasible primal, which together with (approximate) complementary slackness and (approximate) feasibility of φ implies that M is approximately optimal.

Our objective is now to *grow* the matching M up until it is large enough. Without delving into too much detail, it is sufficient to state that the basic operations required to grow M are two. First, we need to compute maximal sets of node-disjoint augmenting paths[5]. Second, we need to compute reachability queries; namely given a set of root vertices R we should output the set of vertices that are reachable from R. The key contribution of this paper is to design the right relaxations of these two basic operations that allow them to be implemented in sublinear time.

Implementing the template algorithm in sublinear time. We have seen that implementing the template algorithm boils down to computing (i) maximal sets of node-disjoint augmenting paths and (ii) reachability queries. The basic ingredient we used in [BR23] to implement both (i) and (ii) is an algorithm from [BKS23] that does the following. Given a unweighted bipartite graph on $2n$ vertices, it returns a matching of size $g(\varepsilon)n$ whenever a matching of size εn exists. Their algorithm runs in time $n^{2-f(\varepsilon)}$[6]. It is not too hard to see that repeatedly applying this algorithm we can find a matching of size $(1-\varepsilon)n$. We define an operation of this kind a *matching query.*

Suppose now that we want to compute a set of node-disjoint augmenting paths using matching queries. If we fix a constant k, a color-coding technique allows us to build $\Omega(n)$ node-disjoint augmenting path of length exactly k in parallel. Thus, using the [BKS23] algorithm we can compute $\Omega(n)$ many node-disjoint shortest paths of *constant* length in time $n^{2-f(\varepsilon)}$. However, there is one caveat: when there are too few node-disjoint augmenting paths the [BKS23] algorithm

[4]Extending this result to arbitrary cost functions taking values in $[0,1]$ is a technicality.

[5]An augmenting path w.r.t. a matching M is a path in the underlying graph where edges alternate between $\in M$ and $\notin M$. Moreover, we require the first and last edges to be $\notin M$. Thus, toggling all edges along an augmenting path increases $|M|$ by one.

[6]Here, $f(\varepsilon), g(\varepsilon) > 0$ are constants that depend on ε.

does not see them. Therefore, it is necessary to relax the notion of maximal set on node-disjoint augmenting paths that we used in the template algorithm to *quasi*-maximal set of *short* node-disjoint augmenting paths.

Likewise, to answer a connectivity query we can build a forest in parallel using matching queries. Here the [BKS23] algorithm fails whenever the forest admits a cut of small size. Therefore, we relax the notion of connectivity to a weaker notion, where we allow to return that a vertex u is not connected to the roots $R \subseteq A \cup B$ as long as there exists a small cut separating u and R.

If we analyse the impact of both these relaxations on the correctness proof of our template algorithm, we realize that only a small fraction of vertices does not satisfy the invariants that hold for the vanilla template algorithm. Hence, labeling these vertices as "outliers" is sufficient to successfully carry out the analysis. Finally, employing the [BKS23] algorithm to compute matching keeps the complexity of our algorithm strictly subquadratic (namely, sublinear in the input size).

2.4 Future Directions

In order to point out interesting future direction, we would like to compare EMD with another well-known metric optimization problem: Minimum Spanning Tree (MST). For both problem we will focus on general metrics.

We would like to point out that there is a separation between the complexity of the (approximate) value version of MST and EMD respectively. In fact, [CS09] designs an algorithm that runs in $\tilde{O}_\varepsilon(n)$ time and returns the *cost* of MST up to a multiplicative factor $1+\varepsilon$. On the other hand, as we already mentioned, no constant approximation for EMD can be computed in $o(n^2)$ time [BCIS05]. Intuitively, this happens because the cost of MST depends on the entries of the distance matrix more *robustly* than the value of EMD. Indeed, increasing a few entries of the distance matrix can alter the value EMD arbitrarily, whereas for MST this does not happen.

However, we showed in [BR23] that allowing additive approximation makes EMD more robust, and thus amenable to sublinear algorithms. We would like to raise as an open problem whether this additional robustness allows, for instance, for an ε-additive approximation algorithm running in $\tilde{O}_\varepsilon(n)$ time, which would match the result for MST [BCIS05].

Chapter 3

Better Sum Estimation via Weighted Sampling

In this chapter, we present the paper "Better Sum Estimation via Weighted Sampling", which was published at SODA 2022 and received the Best Student Paper Award [BR23]. First, we introduce the problem, then we survey some related work and finally describe our technical contribution.

3.1 Introduction

Suppose you are given a set of U of size n, such that each $u \in U$ has a weight $w(u) > 0$, and you are asked to estimate $W = \sum_{u \in U} w(u)$. If n is so large that we cannot afford to scan through U, a natural question is whether we can estimate W in sublinear time. The answer, of course, depends on the access model that we allow. The most natural access model is the one where we can sample from U uniformly. Unfortunately, it is impossible to give any non-trivial estimate of W using $o(n)$ uniform samples, since we could have a single element accounting for 99% of W. The next most natural model is the one where we can sample $u \in U$ with probability proportional to $w(u)$; we dub this access model *proportional* setting.

How many *proportional* samples do we need to output $\hat{W}$ such that $P[\hat{W} \in (1 \pm \varepsilon)W] \geq 2/3$?

Motwani, Panigrahy, and Xu [MPX07] gave an algorithm for the proportional setting using $\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$ queries, as well as a lower bound of $\Omega(\sqrt{n})$ queries. In [BT22], we improve both their upper and lower bound and prove that, up to constant factors, $\sqrt{n}/\varepsilon$ samples are necessary and sufficient. The sample complexity of our new algorithm is thus optimal both in terms of n and ε.

Another natural access model occurs when we allow both proportional and uniform sampling, and we dub this *hybrid* setting. This model was already studied in [MPX07] and in [BT22] we improve their results in this setting as well. Section 3.3 reports our contributions in full detail.

Applications to graph algorithms. The initial problem for a generic set U appears fairly abstract, and it is not clear why it should occur that we are able to sample from U proportionally to $w(\cdot)$. A concrete application for the proportional setting is the following. Suppose we have an extremely large graph $G = (V, E)$, perhaps stored in a distributed fashion, and we would like to estimate $|E|$. If we sample an edge uniformly and pick a random endpoint, this corresponds to sampling a vertex proportionally to its degree. Thus, our algorithm can be employed to estimate

$|E|$ up to a factor $1 \pm \varepsilon$ using $O(\sqrt{|V|}/\varepsilon)$ edge samples. Notice that this improves over the natural birthday-paradox based estimator, that takes $\Theta(\sqrt{|E|})$ samples.

Likewise, if we are allowed to sample uniformly from both V and E, we can use our results for the hybrid setting and obtain fast sublinear algorithms for edge counting.

3.2 Related Work

Besides [MPX07], there is no past work on the sum estimation problem in proportional or hybrid setting. However, similar problems have been considered in the past. Notably, Horvitz and Thompson [HT52] studied how to estimate $\sum_{u\in U} w(u)$ while having sample access to a distribution $\mathcal{D}$ on U, as well as the values $P_{\mathcal{D}}(u)$ and $w(u)$ for each sample $u \sim \mathcal{D}$. In the rest of this section, we will review two lines of research that our paper has an impact on.

First, we review what is known about edge counting in sublinear time. Then, we show how our result can serve as a bridge between proportional sampling and other well-studied sublinear access models.

Edge counting in sublinear time. The problem of estimating simple global properties of a graph G such as $|V|$ or $|E|$ in sublinear time has received significant attention both in theory and practice [Fei06, GR06, Ses15, ERS17, TT21, KLS11, DKS14]. The motive behind all these results is that in practice we encounter graphs G that are too large to scan through, and faster schemes are sought. Not being able to preprocess the graph raises the question about how we the input is represented or, equivalently, what is the access model[1]. Several access models have been explored in the literature for tackling edge counting.

We review how the literature on edge counting in several access models. The algorithm from [Fei06] uses random vertex queries and degree queries[2] and achieves a $(2+\varepsilon)$-approximation using $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}})$ time and queries. Using neighborhood queries[3], Goldreich and Ron showed a $(1+\varepsilon)$-approximation with time and query complexity of $\tilde{O}(\frac{n}{\varepsilon^{9/2}\sqrt{m}})$ [GR06]. The current best algorithm in this setting is by Eden, Ron and Seshadhri and has complexity $\tilde{O}(\frac{n}{\varepsilon^{2}\sqrt{m}})$ [ERS17]. If pair queries[4] along with random vertex and neighbors queries are allowed, Tětek and Thorup designed an algorithm running in time $\tilde{O}(\sqrt{n}/\varepsilon)$ [TT21]. Incidentally, this is the same complexity[5] that we achieve using only random edge and degree queries.

From proportional sampling to distribution testing. Canonne and Rubinfeld considered an access model where the following queries are supported: (i) sample $u \in U$ according to $\mathcal{D}$; (ii) given $u \in U$, return $P_{\mathcal{D}}(u)$ [CR14]. Similarly, Onak and Sun considered a model where one can sample from a distribution $\mathcal{D}$ on U and both $u \in U$ and $P_{\mathcal{D}}(u)$ are returned [OS18]. Both these models are stronger than our proportional setting, indeed they become essentially equivalent to our proportional setting once we know W .

Both these papers solve several distribution testing problems such as: uniformity testing, identity testing, entropy estimation and distance to a known distribution, in the respective models. Moreover, both papers point out that many of their algorithms are robust with respect to perturbations of probability oracles. Therefore, our results serve as a reduction from the proportional (hybrid) setting to these stronger models where the probability mass is returned.

[1]Namely, what set of queries the algorithm is allowed to perform.
[2]Given a vertex v, returns $deg(v)$.
[3]Given a vertex v and an index i, returns the i-th neighbor of v.
[4]Given two vertices u, v, returns $(u, v) \in E$
[5]up to a $\log^{O(1)} n$ factors.

In fact, a multiplicative approximation of W readily translates to approximate probability mass values.

3.3 Our Contribution

Our main contribution is establishing that the complexity of sum estimation in the proportional setting is $\Theta(\sqrt{n}/\varepsilon)$, improving both upper and lower bounds from [MPX07]. As a bonus, our algorithm is a simple formula and thus it is more practical than its predecessors [MPX07]. We achieve a similar result in the hybrid setting, where we show that the complexity is essentially $\Theta(\sqrt[3]{n}/\varepsilon^{4/3})$. See Table 3.1 for a summary of the results in [BT22] and an explicit comparison with [MPX07].

In [MPX07] it was assumed that $n = |U|$ is known. We extend our results to the case of unknown n. We consider the case when we have an upper bound $\tilde{n} \geq n$ as well as when we have no advice on n.

When no advice is given, we design a $O(\sqrt{n}/\varepsilon + \log n/\epsilon^2)$-time algorithm for the proportional setting, thus paying a small price for not having advice. On the other hand, in the hybrid setting we can show a $\Omega(\sqrt{n}/\varepsilon)$ lower bound, which is substantially worse than the complexity with advice. All in all, in the proportional setting we may remove the advice without impacting the complexity much, while in the hybrid setting having an advice gives an advantage. See Table 3.1 for a more direct comparison.

Advice to the algorithm	This paper		Motwani, Panigrahy and Xu [MPX07]	
	Proportional	Hybrid	Proportional	Hybrid
n known	$\Theta(\sqrt{n}/\varepsilon)$	$O(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n\log n))$ $\Omega(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n)$	$\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$ $\Omega(\sqrt{n})$	$\tilde{O}(\sqrt[3]{n}/\varepsilon^{9/2})$, $O(\sqrt{n}/\varepsilon^2)$ $\Omega(\sqrt[3]{n})$
Known $\tilde{n} \geq n$	$O(\sqrt{\tilde{n}}/\varepsilon)$ $\Omega(\sqrt{n}/\varepsilon)$	$O(\min(\sqrt{n}/\varepsilon, n\log n))$ $\Omega(\min(\sqrt{n}/\varepsilon, n))$		
No advice	$O(\sqrt{n}/\varepsilon + \log n/\epsilon^2)$ $\Omega(\sqrt{n}/\varepsilon)$	$O(\min(\sqrt{n}/\varepsilon, n\log n))$ $\Omega(\min(\sqrt{n}/\varepsilon, n))$		

Table 3.1: Results in [BT22].

In the rest of this section, we give an overview of the techniques employed in [BT22].

Proportional setting: algorithm. Here we show how to obtain a simple algorithm (a formula, actually) for sum estimation in the proportional setting.

Sample $u_1, u_2 \in U$ proportionally and let $Y_{12} := 1/w(u_1)$ if $u_1 = u_2$, and $Y_{12} := 0$ otherwise. Thus, we have $E[Y_{12}] = 1/W$. Moreover, estimating $(1 \pm \varepsilon)W$ is equivalent to estimating $(1 \pm \varepsilon)/W$, thus we can just estimate the latter. A straightforward approach is to sample many independent copies of Y_{12}, take the average, and use a concentration bound. Unfortunately, a second-moment analysis shows that we need $\Omega(n/\varepsilon^2)$ independent copies of Y_{12} to have our estimator concentrated in $(1 \pm \varepsilon)/W$ with a constant probability. Instead, we take m samples $u_1, \cdots, u_m$ and consider an estimator Y_{ij} for each pair u_i, u_j for $i \neq j$. This gives us a quadratic speed-up, earning $\binom{m}{2}$ estimators from m samples. The estimators Y_{ij} are not independent, however they are uncorrelated, which is sufficient for a second-moment analysis. Ultimately, this technique reduces the number of samples from $O(n/\varepsilon^2)$ to $O(\sqrt{n}/\varepsilon)$.

We can condense all we wrote above in a simple formula. Let $S = \{u_1, \cdots, u_m\}$ be the set of sampled items, and for $s \in S$ define c_s to be the number of times s is sampled. Then,

$$\hat{W} = \binom{m}{2} \cdot \left(\sum_{s \in S} \frac{\binom{c_s}{2}}{w(s)} \right)^{-1}$$

is a $(1 \pm \varepsilon)$-approximation of W with probability 2/3 as long as $m = \Omega(\sqrt{n}/\varepsilon)$. Notice that choosing a suitable value of m requires to know n or an upper bound $\tilde{n} \geq n$.

When no advice on n is given, we use a bucketing scheme to reduce to a problem where estimating the number of items is feasible[6]. In particular, it is fairly easy to sample a number b such that $B_b = \{u \in U \mid w(u) \in [2^b, 2^{b+1})\}$ contains $\Omega(1/\log n)$ mass in expectation. Then, using rejection sampling we can sample both proportionally and uniformly from B_b. Using a standard collision-based estimator we can estimate $|B_b|$ that, in turn, allows us to use the formula above and estimate $\hat{W}_b \approx W_b = \sum_{u \in B_b} w(u)$. Finally, we estimate $\hat{P}_b \approx P_{prop}(u \in B_b) = W_b/W$ and return $\hat{W} = \hat{W}_b/\hat{P}_b$. The fact that $E_b[P_{prop}(u \in B_b)] = \Omega(1/\log n)$ ensures that rejection sampling only causes a $O(\log n)$ overhead in expectation.

Hybrid setting: algorithm. To solve hybrid setting, we heavily leverage our result for proportional setting. The main ingredient here is to restrict ourselves to the set U_{heavy} of the $n^{2/3}\varepsilon^{2/3}$ items with largest weights and solve sum estimation on U_{heavy} through rejection sampling. If the overhead introduced by rejection sampling is small, then we simply run the algorithm we designed for the proportional setting. Else, the overhead is large, and this implies that the weight of the lightest element in U_{heavy} is comparable to the average weight W/n. Then, the simple estimator $1/w(u)$ for $u \sim P_{prop}|_{U_{heavy}}$ is well-enough concentrated to yield the desired sample complexity.

Notice that the algorithm above uses n explicitly. If we do not have any advice on n, we run a straightforward collision-based estimator for n that takes $O(\sqrt{n})$ uniform samples and return a constant approximation of n. This increases the sample complexity of our algorithm from $\sqrt[3]{n}$ to $\sqrt{n}$. However, as we can see from the lower bounds in Table 3.1, this is inevitable.

Lower bounds. All lower bounds proven in [BT22] follow a common schema that we outline here. First, define a uniform mixture of two instances of sum estimation U_1 and U_2 such that any $(1 \pm \varepsilon)$-approximation to sum estimation distinguishes between them. Then, we show that it statistically impossible to distinguish between them with less than a certain number of samples (depending on the problem at hand).

In order to do so, we prove that if we are able to distinguish between U_1 and U_2 then we must be able to solve at least one of the two following problems.

1. Distinguish between a set of size n and on of size $(1 - \varepsilon)n$ through uniform sampling,
2. Distinguishing between random variables $Bern(p)$ and $Bern(p - \delta)$ through sampling,

for some values of n, ε, δ and p. Then, we leverage statistical lower bounds for these two problem to infer a statistical lower bound for distinguishing between U_1 and U_2.

Problem 2 is know to require $\Omega(p/\varepsilon^2)$ samples, so we do not need to prove anything new. As for problem 1, we sketch how we prove a lower bound on its sample complexity. Let S be the multi-set of uniform samples in problem 1. First, we notice that the ratio $\mathcal{R}(S)$ between the posterior probabilities of the events U_1 and U_2 depends only on the number $\ell(S)$ of distinct

[6]Indeed, using proportional sampling only it is impossible to estimate the total number of items n.

elements in S. Then, we have that, for $i = 1, 2$, $\ell(S) \,|\, U_i$ is concentrated around $E[\ell(S) \,|\, U_i]$. Moreover, $E[\ell(S) \,|\, U_1] \approx E[\ell(S) \,|\, U_2]$. Finally, small variation of $\ell(S)$ produce small variations of $\mathcal{R}(\ell(S))$. These facts are sufficient to infer that $R(\ell(S))$ lies in a narrow interval around 1, with probability ≈ 1. Therefore, the posterior probabilities of U_1 and U_2 are very close and we cannot distinguish between them much better than a random guess.

Chapter 4

Online Packing to Minimize Area or Perimeter

In this chapter, we present the paper "Online Packing to Minimize Area or Perimeter", which was published at SoCG 2021 [AB21]. First, we introduce the problem, then we survey some related work and finally describe our technical contributions.

4.1 Introduction

Issues associated with packing polygons manifest in various major industries. One example is where a specified set of pieces needs to be cut from a large material sheet to minimize waste. This is notably applicable in clothing production, where cutting patterns are extracted from a fabric roll, as well as in sectors such as leather, glass, wood, and sheet metal cutting.

In some applications, pieces come "online" and decisions on placement must be taken before knowing about future pieces. This contrasts with "offline" problems, where all pieces are known in advance. Historically, packing problems were the first test bed for online algorithms when they were first introduced in the 1970s [FW98].

The paper introduced in this chapter [AB21] tackles online packing of rectangles where pieces can be positioned anywhere in the plane without overlapping and the objective is to minimize the size of their axis-parallel bounding box. This size can be measured as the perimeter or the area of the bounding box. Moreover, we can study two variants of this problem where we allow or disallow 90° rotations. The product of these two conditions defines four distinct problems that we tackled in [AB21]: PerimeterRotation, PerimeterTranslation, AreaRotation, and AreaTranslation.

The metric used to evaluate online algorithms is the *competitive ratio*, that is the worst-case ratio between the cost of the given online algorithm and the cost of the best offline algorithm. Essentially, the competitive ratio is an approximation ratio where the algorithm designer has the additional handicap of handling the input online.

4.2 Previous Work

Literature on packing problems is abundant and we defer to surveys for a comprehensive overview [CKPT17, EvS18, vS12, vS15, CW98].

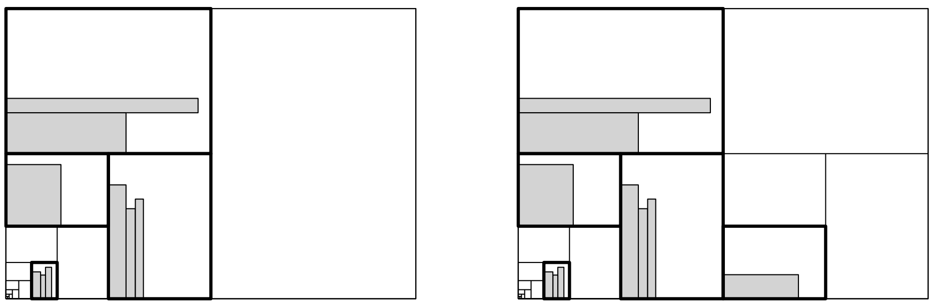


Figure 4.1: Left: Rectangular pieces packed into bricks. Right: A new piece arrives and it is packed in a brand new brick. The brand new brick is picked so that is the smallest that can host the new piece.

Most previous work is concerned with either bin packing (packing the pieces into a minimum number of unit squares) or strip packing (packing the pieces into a $1 \times \infty$ strip so as to minimize the total width of the pieces) [BS83, CW97, YHZ09, HT01, HIYZ07]. Notice that online strip packing and bin packing are at the core topic of Chapter 5 in this thesis, which is a synopsis of [AABK23].

In [AB21], however, we studied a different kind of packing problem where the container is not fixed upfront, we can place pieces anywhere on the place and we aim to minimize the size of the bounding box. This setting was already studied by Alt [Alt16], who showed how a ρ-approximation algorithm for strip packing can be turned into a $(1+\varepsilon)\rho$-approximation algorithm for the offline version of AreaTranslation or AreaRotation, for any constant $\varepsilon > 0$. The main idea in [Alt16] is to tentatively apply the strip packing algorithm to strips of increasing widths and choose the width that results in the smallest area. Clearly, this technique cannot be easily applied in the online setting.

4.3 Our Contribution

In this section we list the contributions in [AB21] and give a sketch of some of the main technical ideas used in their proofs. Table 4.1 summarizes all our results.

Minimizing perimeter. We design online algorithms for PerimeterRotation and PerimeterTranslationand prove that they attain a competitive ratio slightly less than 4.

The main technical idea here is to partition the positive quadrant into *bricks*, which are axis-parallel rectangles with aspect ratio $\sqrt{2}$. Bricks enjoy the property that we can split a brick in half and obtain two identical bricks a factor $\sqrt{2}$ smaller. When a new piece comes, we find the smallest brick that can host it and stack it there. If no brick of the correct size has room for it, a new brick of the correct size is created by partitioning existing bricks. See Figure 4.1 for a graphical representation of our algorithm.

We also give a lower bound of 4/3 for the version with translations and 5/4 for the version with rotations. The proof of these bounds are fairly straightforward and follow a general scheme. First, we feed the online algorithm with a few 1×1 square pieces, then we adaptively choose the next piece between a tall-and-skinny rectangle, a short-and-fat rectangle and a 1×1 square so as to maximize the competitive ratio.

Minimizing area. We study the competitive ratio of AreaRotation and AreaTranslation. We show a lower bound of $\Omega(\sqrt{n})$ on the competitive ratio for both problems. This lower bound is obtained by feeding the online algorithm with n pieces of size $1 \times 1/n$ and then, according to the shape of the current bounding box either a 1×1 square or a $n \times 1/n$ rectangle. If the aspect ratio of the intermediate bounding box is $\leq \sqrt{n}$ then we feed the short-and-fat rectangle, otherwise we feed the square. In both cases we obtain a bounding box which area is $\sqrt{n}$ times larger than the optimal offline solution.

On the algorithmic side, we design algorithms that achieve $O(\sqrt{n})$ competitive ratio for both AreaRotationand AreaTranslation, and are thus optimal up to constant factors. The basic idea behind these algorithms is to use a well-known online shelf-packing algorithm [BS83] using dynamically-allocated boxes as containers.

Minimizing area: special instances. Since we have such strong lower bounds on competitive ratio for the area version of our problem, a natural question is whether we can circumvent these lower bounds by making natural assumptions that invalidate our hard instances. We study two candidates: (i) input pieces have edge length ≥ 1; (ii) input pieces have aspect ratio upper-bounded by $\alpha \geq 1$.

The assumption that all edge length is ≥ 1 does not exactly invalidate the lower bound above. Indeed, it is sufficient to scale each piece by a factor n. However, the area of the bounding box in that instance becomes n^2, which is fairly large. In particular, in terms of the optimal offline solution cost Opt, that same instance gives a $\Omega(\sqrt[4]{OPT})$ lower bound on the competitive ratio. It turns out that we can match that lower bound and give a $O(\sqrt[4]{\text{Opt}})$-competitive algorithm for AreaRotation. Likewise, for AreaTranslationinstead, we prove that $\Theta(\sqrt{\text{Opt}})$ is the right competitive ratio in terms of Opt.

This might suggest that we should investigate competitive ratio as a function of Opt. However, when the pieces can be arbitrary, no function of Opt can bound the competitive ratio for AreaRotation nor AreaTranslation.

As for (ii), we study the case $\alpha = 1$ (namely, all pieces are squares) and extend to general α subsequently. A similar problem had already been studied: online packing of squares into a square bounding box [FH17a]. There, an 8-competitive algorithm was given. In [AB21], we improve the analysis from [FH17a] and show that their algorithm is in fact 6-competitive and that this is tight. Given such result, it easily follows that for any constant bound $\alpha \geq 1$ on the aspect ratio, if the goal is to minimize the area of the axis-parallel bounding rectangle, we also get a $O(1)$-competitive algorithm.

Measure	**Version**	**Trans./Rot.**	**Lower bound**	**Upper bound**
Perimeter	General	Translation	4/3	3.98
		Rotation	5/4	3.98
Area	General	Translation	$\Omega(\sqrt{n})$ & $\forall f : \Omega(f(\textsc{Opt}))$	$O(\sqrt{n})$
		Rotation	$\Omega(\sqrt{n})$ & $\forall f : \Omega(f(\textsc{Opt}))$	$O(\sqrt{n})$
	Sq.-in-sq.	N/A	16/9	6
	Edge length ≥ 1	Translation	$\Omega(\sqrt{\textsc{Opt}})$	$O(\sqrt{n}) = O(\sqrt{\textsc{Opt}})$
		Rotation	$\Omega(\max\{\sqrt{n}, \sqrt[4]{\textsc{Opt}}\})$	$O(\min\{\sqrt{n}, \sqrt[4]{\textsc{Opt}}\})$

Table 4.1: Results in [AB21]. Each entry refers to the competitive ratio of the cooresponding problem.

Chapter 5

Online Sorting and Translational Packing of Convex Polygons

In this chapter, we present the paper "Online Sorting and Translational Packing of Convex Polygons", which was published at SODA 2023 [AABK23]. First, we introduce the problem, then we survey some related work and describe our technical contributions. In the end, we point out some interesting future directions.

5.1 Introduction

Imagine that you are given an infinitely long roll of fabric (Figure 5.1) and you are asked to cut some convex pieces of fabric out of this roll so as to minimize the amount of wasted fabric. Assume that the requests for convex pieces come online, namely you have to deliver them as they come. Moreover, the fabric has a privileged direction and every piece request comes with a specific orientation (or, equivalently, you are not allowed to rotate the pieces). We studied this and other related problems in [AABK23].

More formally, in [AABK23] we studied several online packing problems on the plane, where the input is a sequence of convex polygons that have to be irrevocably placed in a given container as they come, without rotating them. If the container is the infinite strip describe above we talk about Strip-Packing and this will be the container that we consider throughout, unless otherwise specified.

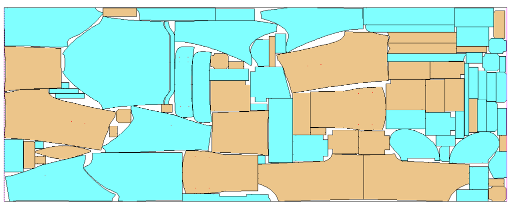
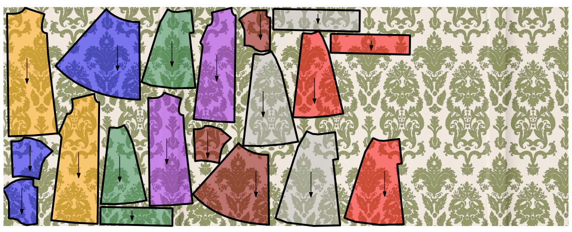



Figure 5.1: Left: Real-world example of packing on fabric produced using industrial software from Mirisys. Right: Potential sewing patterns for the clothes arranged on roll of fabric.

For strip packing, if we require pieces to be cut online but we relax the constraint on rotations and allow pieces to be rotated, a $O(1)$-competitive[1] algorithm exists [BS83]. Likewise, if retain the no-rotation constraint but we allow for an offline solution an efficient $O(1)$-approximation algorithm exists [AdBK17]. Therefore, it would be natural to conjecture that a $O(1)$-competitive online algorithm exists also when we are not allowed to rotate pieces. Surprisingly, in [AABK23] we proved a superconstant lower bound on the competitive ratio for online strip packing without rotations.

The key idea behind our proof is a reduction to a combinatorial problem that we dub *online sorting*. Essentially, we show that packing convex polygons online is costly because we cannot sort them by slope online. Formally, we prove that if we had a $O(1)$-competitive online algorithm for packing, then we would have a $O(1)$-competitive algorithm for online sorting. Then, we rule out the latter with an ad-hoc argument.

It is interesting to observe that the known efficient offline $O(1)$-approximation algorithm for packing sort convex pieces by slope [AdBK17]. In [AABK23] we proved that sorting is, in some sense, a necessary step to obtain a constant-factor approximation.

Packing problems studied. In [AABK23], we studied several packing problems. In Strip-Packing, we have a horizontal strip of height 1 which is bounded to the left by a vertical segment and unbounded to the right. The goal is to place the pieces so that they occupy a prefix of the strip of minimum length. In Bin-Packing, the pieces have to be placed in unit squares, and the goal is to use a minimum number of squares. In Perimeter-Packing, pieces are placed in the plane, and the goal is to minimize the perimeter of their axis-parallel bounding box, as in Chapter 4.

When the pieces are restricted to axis-parallel rectangles, there are online algorithms with constant competitive ratios solving all the problems above. Indeed, any online algorithm that works for axis-parallel rectangular pieces extends to arbitrary convex polygon if these can be rotated, because it is always possible to rotate a convex piece so that it has density $1/2$ in its axis-parallel bounding box. Thus, all the problems above admit constant-competitive algorithm if we lift the no-rotation constraint.

In [AABK23] we prove that all the problems above admit efficient $O(1)$-approximation offline algorithm (through reductions to [AdBK17]).

A surprising lower bound. Even though all the packing problems above admit $O(1)$-competitive online algorithms when rotations are allowed and efficient $O(1)$-approximation offline algorithms, in [AABK23] we prove superconstant lower bounds on the competitive ratio of all the problems above, when rotations are forbidden.

5.2 Previous Work

As pointed out in Chapter 4, the literature on online packing problems is vast and we point the reader to more comprehensive surveys [CKPT17, EvS18, vS12, vS15, CW98]. Below we survey the most important results related to the packing problems studied in this paper.

Strip packing. If the pieces are simple or convex polygon, exact and efficient approximation offline algorithms for strip packing have been studied [Mil96, Mil97, MD99, DM97]. If the pieces are rectangular, constant-competitive algorithms have been studied [BS83, YHZ09, CW97, HIYZ07].

[1]The notion of competitive ratio is defined in Chapter 4.

Bin packing. Online bin packing problems have been studied since the early 1970s [FW98] and have seen a significant amount of work. A long sequences of papers, brought the upper bound on the asymptotic competitive ratio[2] for this problem down to 2.5545 and the lower bound up to to 1.907 [HCT+11].

Perimeter packing. The offline versions of this and similar problems has received significant attention [Mil96, Mil99, MD99, AC12, AH00, LW88, PBAA16, Alt16, AdBK17, LG03, Spe13, LG09, EG75, CG19] and it has been shown that perimeter packing admits efficient $O(1)$-approximation algorithms. The online version has received less attention, nonetheless it has been proven that if the polygons to be packed are rectangles then $O(1)$-competitive algorithms exist [FH17b, AB21].

5.3 Our Contribution

Our main innovative contribution is the definition of an auxiliary problem, Online-Sorting, that serves as a stepping stone for our lower bound. First, we reduce strip packing to online sorting and then reduce all our packing problems to strip packing.

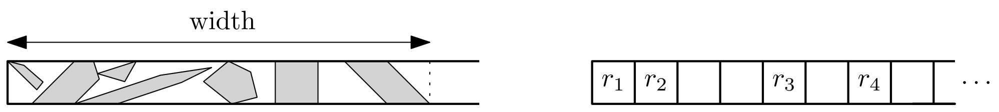


Figure 5.2: Strip packing and online sorting.

Online sorting. In Online-Sorting$[\gamma, n]$, we have an empty array A with $\lfloor \gamma n \rfloor$ cells for some $\gamma \geq 1$, and receive a stream of real numbers $s_1, \ldots, s_n$, $s_i \in [0, 1]$. Each real has to be placed into an empty cell of A before the next real is known. The objective is to minimize the sum of differences of consecutive reals in A, where $A[i]$ and $A[j]$ are consecutive if $i < j$ and $A[i+1], \ldots, A[j-1]$ are all empty. The offline optimum is obtained by placing the reals in sorted order in some n cells of A, hence the name "online sorting". We show that Online-Sorting$[O(1), n]$ does not allow for constant factor competitive online algorithms, and more precisely we prove the following.

Theorem (Theorem 1 in [AABK23]). *Suppose that $\gamma, \Delta \geq 1$ are such that Online-Sorting$[\gamma, n]$ admits a Δ-competitive algorithm, then $\gamma\Delta = \Omega(\log n / \log \log n)$.*

In order to prove the theorem above, we need to design an adversarial input that is chosen adaptively with respect to the algorithm's placements. At a high level, we would like to force the algorithm to place far numbers in adjacent positions. We design an adversary that operates in rounds such that at round i it outputs reals from a given set S_i. We design sets S_i so that they are nested ($S_{i+1} \subseteq S_i$) and we define S_{i+1} adaptively so that there exists a set D of *deserted cells* of A such that either: (i) the algorithm cannot place many reals from S_{i+1} or (ii) the algorithm incurs a high cost. In both cases we make progress, indeed in case (i) we have virtually decreased the factor γ (i.e., the memory slack) and in case (ii) we have come closer to our goal of increasing the competitive ratio.

[2]A notion of competitive ratio that only looks at worst-case instances of sufficiently large size.

Reducing online packing to online sorting. A fundamental step in proving our lower bound is to reduce all our packing problems to online sorting. Online strip packing is the most natural problem to reduce to online sorting. It is fairly easy to see that, if we restrict ourselves to parallelogram pieces which height is exactly the height of the strip, then the two problems are related. See Figure 5.3.

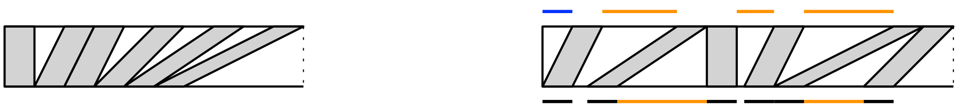


Figure 5.3: Two strip packings of the same set of parallelograms which height is the same of the strip's height. Left: An optimal packing is obtained through sorting. Right: An arbitrary packing creates some gaps. These gaps are proportional to the overhang of parallelograms, that we translate to reals in our reduction.

This instance with uniform-height parallelograms might seems contrived, and one may wonder whether packing is still hard when we restrict ourselves to pieces that have diameter $\leq \delta$ for some arbitrarily small $\delta > 0$. The same lower bound holds even under such restriction, and it extends to all studied packing problems too. See Figure 5.4 for a map of our reductions.

$$\begin{tikzcd}
 & & \boxed{\textsc{Strip-Packing}, \varnothing \leq \delta} & \\
\boxed{\textsc{Online-Sorting}[C, n]} \arrow[r] & \boxed{\textsc{Strip-Packing}} \arrow[ur] \arrow[r] \arrow[dr] & \boxed{\textsc{Bin-Packing}, \varnothing \leq \delta} & \\
 & & \boxed{\textsc{Square-Packing}[\delta]} \arrow[r] & \boxed{\textsc{Perimeter-Packing}}
\end{tikzcd}$$

Figure 5.4: An overview of our reductions. Note that an arrow from problem A to problem B means that an algorithm for B implies an algorithm for A. Here, $\varnothing \leq \delta$ means that the diameter of each piece is at most an arbitrary constant $\delta > 0$. The problem $\textsc{Square-Packing}[\delta]$ is similar to bin packing and we refer the reader to the full version of [AABK23] for its definition.

The following theorem summarizes most of the lower bounds for packing in [AABK23].

Theorem (Essentially Theorem 2 in [AABK23])**.** *Fix $\delta > 0$. Given a stream of n pieces, which are restricted to have diameter $\leq \delta$:*

1. *The competitive ratio or* STRIP-PACKING *is $\Omega(\sqrt{\log n / \log\log n})$.*
2. *The competitive ratio or* BIN-PACKING *is $\Omega(\sqrt{\log n / \log\log n})$.*
3. *The competitive ratio of* PERIMETER-PACKING *is $\Omega(\sqrt[4]{\log n / \log\log n})$.*

Algorithms. We complement our lower bounds with two algorithmic results.

- For $\textsc{Online-Sorting}[1+\varepsilon, n]$, we give an online algorithm with competitive ratio $2^{O\left(\sqrt{\log n} \cdot \sqrt{\log\log n}\right)}$.
- For strip packing, we give an online algorithm with competitive ratio $O(n^{\log 3 - 1} \log n)$.

Both algorithms use recursive schemes to achieve non-trivial competitive ratios. We believe that these algorithms are of minor interest and their main goal was to initiate the study of the algorithms as well. Indeed, we would expect it to be possible to achieve much lower competitive ratios than the ones achieved in [AABK23].

For both online sorting and strip packing there is an exponential gap between the lower and upper bound on the competitive ratio.

5.4 Future Directions

Two main future directions were left open in [AABK23]. First, the design of our adversary was completely adaptive. This implies that our lower bound does not work against randomized algorithms, unless the adversary has white-box access to the algorithm's internal state. Therefore, a natural question is whether there exists a randomized non-adaptive adversary, which would work also against randomized online algorithms.

Second, there is an exponential gap between the lower and upper bounds on the competitive ratio of both strip packing and online sorting. We believe that an extremely natural question is to close this gap.

Chapter 6

Locally Uniform Hashing

In this chapter, we present the paper “Locally Uniform Hashing”, which was published at FOCS 2023 [BBK$^+$23]. First, we introduce the problem, then we survey some related work and finally describe our technical contributions.

6.1 Introduction

Hashing is a ubiquitous technique in computer science, with a strong impact on the resources spent on computation. In theory, algorithms are often analyzed under the unrealistic assumption that a fully random hash function exists. Such assumption is indeed unrealistic, because a fully random hash function $h : U \to \mathcal{R}$ uses $|U| \log |\mathcal{R}|$ bits of space, and the universe U is often prohibitively large. In practice, fast hash functions with weak theoretical guarantees are employed. The goal of [BBK$^+$23] is to develop a practical hash function that provably behaves as if it was (almost) fully random.

Using weak hash functions can have catastrophic consequences. A prominent example is the classic linear hashing data structure, implemented with a 2-independent hash function[1]. If the input carries enough entropy, then any 2-independent hashing performs well [MV08]. However, if the input is structured (e.g., an arithmetic sequence) certain 2-independent hash families perform poorly [TZ12, PT10]. This issue becomes even worse if we shift from data structures to streaming computation, where it is not even possible to detect an anomaly because our system does not store the original data. In this setting, our estimators could be flawed without us realizing it. Only a hash function with worst-case guarantees can prevent such failures.

In [BBK$^+$23], we develop *tornado tabulation*, a hash function that satisfies a certain local full randomness property. Local full randomness ensures that tornado tabulation can be used as a plug-and-play replacement for the (unrealistic) fully random hash function in many analyses of algorithms. In turn, this allows for a practical implementation of such algorithms that is still theoretically sound. Examples of these applications are the classic linear probing scheme [Knu63], HyperLogLog [FFGM07] and One-Permutation Hashing [LOZ12].

6.2 Previous Work

Tabulation-based hashing has received significant attention in the past 15 years [HT22, DKRT14, DT14, Tho13, PT13, PT12a, KW12, TZ12, TZ04, TZ09, PT12b]. On a practical standpoint,

[1]A hash function is k-independent if its restriction to any set of keys of size k is fully random.

tabulation-based hashing is remarkably fast because it uses a few simple bit-wise operation and small tables that fit in fast cache. On a theoretical standpoint, even though tabulation hashing is not even 4-independent, it works great inside popular algorithms [PT12b]. Unfortunately, these applications require ad-hoc proofs showing that tabulation hashing does work. In [BBK$^+$23], we prove that tornado tabulation satisfies a more general local full randomness property that allows it to be used inside popular algorithms more like a black box.

An enhanced notion of full randomness. Our notion of local full randomness is inherited[2] from [DKRT15] and roughly speaking it says that if we restrict ourselves to a small portion of the hash table, the elements hashed there are hashed (almost) uniformly at random. This formulation is sufficient for certain applications but does not work when the interesting portion of the hash table depends on the hash values of some query keys.

For instance, in the analysis of linear probing, given a query key q, we would like to bound the number of contiguously occupied cells in the sequence $h(q), h(q)+1, \dots$ known as *run length.* In fact, the time spent to lookup (or insert) q is exactly proportional to the run length starting at $h(q)$. For our analysis to carry over as if we had a fully random hash function we need to guarantee that the set of keys hashed in a sizeable interval around q are hashed at random.

In [BBK$^+$23], we enhance their notion of local full randomness so that the small portion of the hash table considered interesting is allowed to depend on the hash values of some query keys. The notion of local full randomness is rather technical in itself, and we present it in the next section.

6.3 Our Contribution

In this section we formally define tornado tabulation hashing and present the main theorem in [BBK$^+$23].

Simple tabulation hashing. We assume that keys are c-tuples of *characters* $(x_1 \dots x_c) \in \Sigma^c$ and we map them to hash values in $\mathcal{R}$. We then say $h : \Sigma^c \longrightarrow \mathcal{R}$ is a *simple tabulation* hash function if

$$h(x) = T_1[x_1] \oplus \cdots \oplus T_c[x_c]$$

where, for each $i = 1 \dots c$, $T_i : \Sigma \longrightarrow \mathcal{R}$ is a fully-random function stored as a table.

Tornado tabulation hashing. To define a tornado tabulation hash function h, we use several simple tabulation hash functions as building blocks. A tornado tabulation function has a number d of *derived* characters.

For each $i = 0, \dots, d$, we let $\widetilde{h}_i : \Sigma^{c+i-1} \longrightarrow \Sigma$ be a simple tabulation hash function. Given a key $x \in \Sigma^c$, we define its *derived key* $\widetilde{h}(x) \in \Sigma^{c+d}$ as $\widetilde{x} = \widetilde{x}_1 \cdots \widetilde{x}_{c+d}$, where

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i < c \\ x_c \oplus \widetilde{h}_0(\widetilde{x}_1 \cdots \widetilde{x}_{c-1}) & \text{if } i = c \\ \widetilde{h}_{i-c}\,(\widetilde{x}_1 \cdots \widetilde{x}_{i-1}) & \text{if } i > c. \end{cases} \tag{6.1}$$

We note that each of the d derived characters $\widetilde{x}_{c+1}, \dots, \widetilde{x}_{c+d}$ is progressively defined by applying a simple tabulation hash function to all its preceding characters in the derived key $\widetilde{x}$. Finally, we have a simple tabulation hash function $\widehat{h} : \Sigma^{c+d} \longrightarrow \mathcal{R}$, that we apply to the derived key. The *tornado tabulation* hash function $h : \Sigma^c \longrightarrow \mathcal{R}$ is then defined as $h(x) = \widehat{h}(\widetilde{x})$.

[2]Although they did not name this property explicitly.

Implementation of tornado tabulation. The simplicity of tornado tabulation is apparent from its C implementation below. An important low-level optimization is folding tornado's tables together so that it only performs one lookup per character, hence $c + d$ lookups in total. We implement these folded tables as $c+d$ character tables $\Sigma \to \Sigma^{d+1} \times \mathcal{R}$, where elements of $\Sigma^{d+1} \times \mathcal{R}$ are just represented as w-bits numbers.

The C-code implementation below works for 32-bit keys, with $\Sigma = [2^8]$, $c = 4$, $d = 4$, and $\mathcal{R} = [2^{24}]$. Besides the key x, the function takes as input a randomly initialized array of $c + d$ tables of size $|\Sigma|$. It is worth to notice that the size of Σ is chosen so that a table of size $|\Sigma| \cdot d \cdot \log |\mathcal{R}|$ fits in fast cache.

```
INT32 Tornado(INT32 x, INT64[8][256] H) {
          INT32 i; INT64 h=0; INT8 c;
          for (i=0;i<3;i++) {
                    c=x;
                    x>>=8;
                    h^=H[i][c];}
          h^=x;
          for (i=3;i<8;i++) {
                    c=h;
                    h>>=8;
                    h^=H[i][c];}
          return ((INT32) h);}
```

Full randomness on query-selected keys. The main result of [BBK+23] is fairly technical. We recall that the goal here is to show that if we select a small set X of relevant keys they have uniformly random hash values. Moreover, whether a given key x is selected or not can depend on: x, $h(x)$ and $h|_Q$, namely the value of h on all query keys in Q.

An hash value $h(x) \in \mathcal{R}$ is described using $\log |\mathcal{R}|$ bits. If we think about a standard representation, the upper bits identify the dyadic interval that $h(x)$ belongs to whereas the lower bits identify its position inside such dyadic interval. If we are interested to select elements that are hashed in a certain interval around $h(q)$ for some $q \in Q$ we do not really care about the lower bits of $h(x)$ nor $h|_Q$, but we do care about the upper bits of $h(x)$ and $h|_Q$. On the other hand, when we care about $h(x)$ being random within a certain interval, we only care about its lower bits. This, motivates the following technical definition.

First, we identify $\mathcal{R} = [2^s] \times [2^t]$, where s are the upper bits (used to select keys) and $[2^t]$ are the lower bits (that we would like to be random). Formally, we define a selector function $f : \Sigma^c \times [2^s] \times [2^s]^Q \longrightarrow \{0, 1\}$ and we define the set of selected keys as

$$X^{f,h} = \left\{ x \in \Sigma^c \mid f(x, h^{(s)}(x), h^{(s)}|_Q) = 1 \right\}$$

where $h^{(s)}(x)$ represents the upper bits of $h(x)$. We then define

$$\mu := \sum_{x \in \Sigma^c} p_x^f \quad \text{with} \quad p_x^f := \max_{\varphi \in \mathcal{R}^Q} \Pr_{r \sim \mathcal{U}(\mathcal{R})} [f(x, r, \varphi) = 1] \ ,$$

where the maximum is taken among all possible assignments of hash values to query keys $\varphi : Q \to \mathcal{R}$ and r is distributed uniformly over $\mathcal{R}$. To help making sense of the expressions above, it is worth to notice that if $h : \Sigma^c \to \mathcal{R}$ is fully random then $\mathrm{E}[|X^{f,h}|] \leq \mu$.

Now, we can give our main result

Theorem (Theorem 5 in [BBK$^+$23]). *Let $h : \Sigma^c \to \mathcal{R}$ be a tornado tabulation hash function with d derived characters and f be a selector function operating on the s upper bits as described above. If $\mu \leq \Sigma/2$, then the t lower bits of h are fully random on $X^{f,h^{(s)}}$ with probability at least*

$$1 - (7\mu^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}) \ .$$

In a nutshell, the above theorem states that if we select our interesting keys based on the upper bits of their hash values, then the lower bits of their hash values are fully random with high probability. This technical condition on upper/lower bits is inevitable, because the upper bits of a given key could be completely determined by the upper bits of the query keys, if these are chosen carefully.

It is worth to notice that, unlike [DKRT15], we compute the constants in our failure probability, which ensures applicability also when Σ is moderately large (and thus fits in cache).

The proof of the theorem above is definitely technical and we point the reader to the full version of the paper for an overview. The main intuition behind it is that subsequent applications of simple tabulation help "breaking" dependencies among keys. Indeed, one can observe that as the number d of derived character increases the failure probability decays exponentially because more and more (random) dependencies are broken.

Chapter 7

Multi-Swap k-Means++

In this chapter, we present the paper "Multi-Swap k-Means++", which was published at NeurIPS 2023 [BCALP23]. First, we introduce the problem, then we survey some related work and describe our technical contributions. In the end, we point out some interesting future directions.

7.1 Introduction

Clustering is a fundamental problem in unsupervised learning where the objective is to partition datapoints into clusters so that similar points belong to the same cluster and dissimilar points do not. Among the wealth of notions of clustering that have been proposed in machine learning, here we focus on one popular formulation: Euclidean k-means.

The definition of Euclidean k-means is the following. Given a dataset P in $\mathbb{R}^d$ we want to find a set of k centers C so that the sum over $p \in P$ of the square Euclidean distances between p and its closest center in C is minimized. Formally, we want to find C minimizing $\sum_{p \in P} \min_{c \in C} ||p - c||^2$.

In [BCALP23], we develop a new seeding strategy for Euclidean k-means that strikes a new balance between theoretical guarantees and practicality.

7.2 Related Work

In this section we introduce the line of research on seeding algorithms for k-mean that [BCALP23] contributed to.

Lloyd's algorithm. The popularity of the k-means objective is partially due to the existence of an extremely simple and effective heuristic for k-means, the so-called Lloyd's algorithm [Llo57]. Lloyd's initializes a set C of k centers arbitrarily, and then it alternates between two steps. First, it assigns each point in P to its closest center in C defining a partition of P into clusters $C_1 \dots C_k$. Second, it updates C so that $C = \{mean(C_i) \mid i = 1 \dots k\}$. Despite its great practical performance, [AV09] showed that Lloyd's can lead to solutions with arbitrarily high approximation ratio. However, it is easy to verify that none of the steps of Lloyd's can increase the objective function $\sum_{p \in P} \min_{c \in C} ||p - c||^2$. Thus, if a seeding strategy (namely, a way to initialize the centers) has a provable approximation guarantee, then this guarantee is inherited by Lloyd's, if initialized with that strategy.

k-means++. An extremely simple strategy of this kind exists, it is known as k-means++ and it gives a $O(\log k)$ approximation factor [AV07]. The way it works is the following. We sample the first center c_1 uniformly at random from P, then for $i = 2 \dots k$ we sample c_i from P so that $x \in P$ is sampled with probability proportional to $\min_{j<i} ||x - c||^2$. In other words, each new center is sampled from P with probability proportional to its current cost.

Local search k-means++. The extreme simplicity of k-means++ led it to be widely adopted and be part of the popular scikit-learn library [PVG$^+$11]. Following this successful application of theory, [LS19b, CGPR20] showed that adding roughly k steps of local search on top of k-means++ yields a still-practical seeding strategy achieving constant approximation.

The local-search step [LS19b, CGPR20] is the following. Given a set of k centers C, a new center c_{k+1} is sampled from P according to the k-means++ distribution, that is $x \in P$ is sampled with probability proportional to $\min_{c \in C} ||x - c||^2$. Then, a center $c_j \in \{c_1 \dots c_{k+1}\}$ is evicted. In particular, we evict the center c_j that, once evicted, makes the current cost increase by the least amount. This local search strategy yields a constant approximation, however this constant is very large (~ 500) and the question on whether this could be improved remained open.

The complexity of approximating Euclidean k-means. From a theoretical standpoint, Euclidean k-means is hard to solve exactly and even hard to approximate [GI03, ACKS15]. Indeed, a long line of research showed that computing a 1.06-approximation to Euclidean k-means is NP-hard [LSW17, CK19, CLK22, CKL21]. On the algorithmic side, [CEMN22] recently improved over a long line of work [KMN$^+$04, JV01, ANFSW19, GOR$^+$22] and obtained a 5.912-approximation for Euclidean k-means in polynomial time.

Primal-dual vs local search. The approximation algorithms in the literature achieving the lowest approximation ratios are based on a reduction to facility location together with primal-dual schema to approximate the latter, and they are arguably impractical [JV01, ANFSW19, GOR$^+$22, CEMN22, CEMN22]. On the other hand, the algorithm in [KMN$^+$04] uses a simple local search strategy. In [BCALP23], we give an alternate implementation of the general local search strategy in [KMN$^+$04] and obtain a more practical algorithm.

Exhaustive local search. Here we review the local search strategy from [KMN$^+$04]. We fix a parameter p that defines the size of our neighborhood in the local search and initialize a set of $C \subseteq F$ centers arbitrarily.

We define a local search step as follows. If there is a set C^+ of at most p points in $\mathbb{R}^d$ together with a set C^- of $|C^+|$ points in C such that $C \setminus C^- \cup C^+$ achieves a better k-means cost than C, then set $C \leftarrow C \setminus C^- \cup C^+$. We run local search steps until convergence.

The local search strategy above is fairly straightforward, however establishing whether there exists a set C^+ of points in $\mathbb{R}^d$ that could be swapped with elements of C to improve the cost is nontrivial. In [KMN$^+$04] this was done restricting the search to an ε-grid of ε^{-d} discrete points and searching exhaustively among them, which gives an undesirable exponential dependency in the dimension d.

To improve the exponential dependency of the exhaustive local search in [KMN$^+$04], we can use coresets [CLSS22] together with dimensionality reduction techniques [BBC$^+$19, MMR19]. This restricts the search for new centers to a set of size $k^{O(\varepsilon^{-2}\log(1/\varepsilon))}$. Thus, a single local search step takes $k^{O(p\varepsilon^{-2}\log(1/\varepsilon))}$ time.

Moreover, setting $p = \Theta(1/\varepsilon)$ suffices to obtain a $(9+\varepsilon)$-approximation [KMN$^+$04]. Therefore, the local search step described in [KMN$^+$04], combined with the latest coreset and dimensionality reduction technology gives a running time of $k^{O(\varepsilon^{-3}\log(1/\varepsilon))}$ per step and achieves a $(9+\varepsilon)$-approximation. Notice that 9 is the best approximation ratio achievable by an algorithm performing such local search, for any constant p [KMN+04].

7.3 Our Contribution

In [BCALP23], we interpolate between the local search strategies of [LS19a] and [KMN+04] and achieve the following results.

- First, we adapt techniques from the analysis of [KMN+04] to obtain a tighter analysis of the algorithm in [LS19b]. In particular, we show that their algorithm achieves an approximation of ratio of ≈ 26.64, whereas the previous analyses [LS19b, CGPR20] reported an approximation ratio of 500.
- Second, we extend the local search step from [LS19a] to multi-swaps, where we swap more than one center simultaneously. This yields a 10.48-approximation in time $O(nd \cdot poly(k))$.
- Third, borrowing techniques from [CASS21], we further improve our local search step and bring the approximation ratio down to $9+\varepsilon$, which is tight for local-search algorithms [KMN+04]. Moreover, this matches the approximation achieved by [KMN+04] while being more efficient. Indeed, we improve the complexity of a local search step from $k^{O(\varepsilon^{-3}\log(1/\varepsilon))}$ to $k^{O(\varepsilon^{-2}\log(1/\varepsilon))}$.

Finally, we provide experiments where we compare a variant of our algorithm against k-means++ and [LS19b].

Our Algorithm. Here we present our algorithm. Let p be the swap size. First, we initialize a set of centers C using k-means++. Then, we run $O(ndk^{p-1})$ rounds of local search, where each local search round works as follows.

We sample a set $In = \{q_1 \dots q_p\}$ of points from P independently so that each point in $x \in P$ is sampled with probability proportional to its cost with respect to C, $\min_{c \in C} ||x-c||^2$. Then, we iterate over all possible subsets $Out = \{c_1 \dots c_p\}$ of $P \cup In$ of size p and select the set Out such that performing the swap (In, Out) maximally improves the cost. If swapping In and Out actually improves the cost, then we do swap In and Out, else we do not perform any swap in this round.

Our algorithm satisfies the following.

Theorem (Theorem 3 and Corollary 5 in [BCALP23]). *For any $\delta > 0$, the p-swap local search algorithm above runs in $\widetilde{O}(ndk^{2p})$ time and, with constant probability, finds an $(\eta^2+\delta)$-approximation of k-means, where η satisfies*

$$\eta^2 - (2+2/p)\eta - (4+2/p) = 0.$$

Thus, for $p = O(1)$ large enough, multi-swap local search achieves an approximation ratio < 10.48 in time $O(nd \cdot poly(k))$.

Notice that setting $p=1$ retrieves exactly the algorithm from [LS19a, CGPR20], which proves that their algorithm actually achieves an approximation ratio < 26.64.

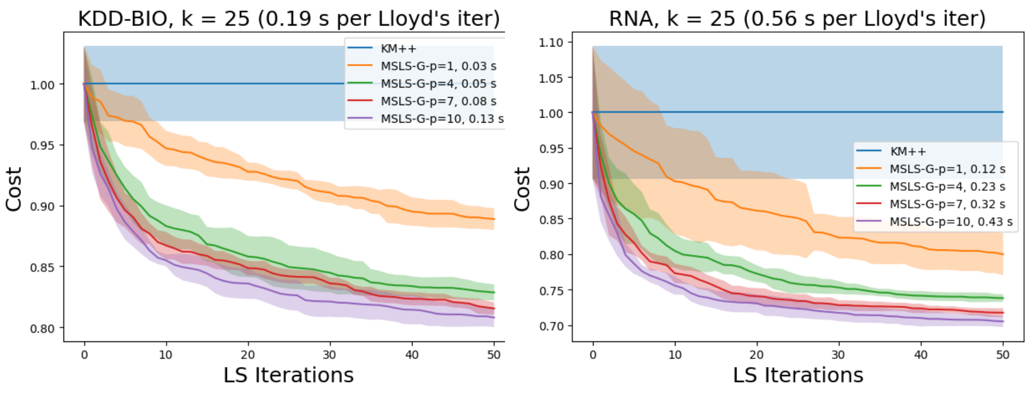


Figure 7.1: This figure compares the cost of our p-swap local search algorithm (MSLS-G), for $p \in \{1, 4, 7, 10\}$, divided by the mean cost of k-means++ (KM++) at each local search step, for $k = 25$. The legend reports also the running time of MSLS-G per local search step (in seconds).

Matching the local-search barrier. As we mentioned above, combining our techniques with those of [CASS21] we manage to match the best approximation achievable by local search algorithm. At a high level, the limitation we found while trying to match the local-search barrier was that our algorithm as described so far is only allowed to select centers from the original dataset P. To overcome this, we needed to consider centers that are convex combination of datapoints, and not just datapoints.

This is exactly what was done in [CASS21], where it was shown that taking the average of a constant number of samples is sufficient to retrieve a good center. We integrated their techniques into our framework and obtained a $(9 + \varepsilon)$-approximation. Although this version of the algorithm is highly impractical, it was an interesting proof-of-concept to see that it is faster than exhaustive local search, even when we enhance the latter with all the latest coreset and dimensionality reduction machinery.

Experiments. In [BCALP23], we run experiments on a greedy variant[1] of our algorithm and showed that its performance improves as the number p of centers swapped simultaneously increases. Figure 7.1 provides an example of our experimental work.

In particular, our approach outperforms previous seeding strategies: k-means++ [AV07] and single-swap local search [LS19b] on popular datasets. See [BCALP23] for a full description of the experimental setting.

7.4 Future work

An interesting open question is to improve the complexity of our local search step. This should be done in two ways. First, we should avoid performing exhaustive search over all possible size-p subsets of centers to swap out. Second, we should prove that fewer than $O(k^{2p})$ repetitions are sufficient to sample a good candidate set of centers to swap in. If both goals are achieved, this would eliminate the term $k^{poly(\varepsilon)}$, leaving hope for an algorithm with running time $\tilde{O}(f(1/\varepsilon)ndk)$ also known as efficient PTAS.

[1]This greedy variant replaces the exhaustive search for the set of centers to swap out with a greedy procedure that chooses centers to swap out one by one. See [BCALP23] for a detailed description of this more practical variant.

Bibliography

[AABK23] Anders Aamand, Mikkel Abrahamsen, Lorenzo Beretta, and Linda Kleist. Online sorting and translational packing of convex polygons. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1806–1833. SIAM, 2023.

[AB21] Mikkel Abrahamsen and Lorenzo Beretta. Online packing to minimize area or perimeter. In *37th International Symposium on Computational Geometry*, 2021.

[AC12] Hee-Kap Ahn and Otfried Cheong. Aligning two convex figures to minimize area or perimeter. *Algorithmica*, 62(1-2):464–479, 2012.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[ACKS15] Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of euclidean k-means. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, volume 34 of *LIPIcs*, pages 754–767. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[ACRX22] Pankaj K Agarwal, Hsien-Chih Chang, Sharath Raghvendra, and Allen Xiao. Deterministic, near-linear $(1+\epsilon)$-approximation algorithm for geometric bipartite matching. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1052–1065, 2022.

[ADBIW09] Alexandr Andoni, Khanh Do Ba, Piotr Indyk, and David Woodruff. Efficient sketches for earth-mover distance, with applications. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 324–330. IEEE, 2009.

[AdBK17] Helmut Alt, Mark de Berg, and Christian Knauer. Approximating minimum-area rectangular and convex containers for packing convex polygons. *JoCG*, 8(1):1–10, 2017.

[AH00] Helmut Alt and Ferran Hurtado. Packing convex polygons into rectangular boxes. In *18th Japanese Conference on Discrete and Computational Geometry (JCDCGG 2000)*, pages 67–80, 2000.

[AIK08] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *SODA*, volume 8, pages 343–352, 2008.

[Alt16] Helmut Alt. Computational aspects of packing problems. *Bulletin of the EATCS*, 118, 2016.

[ALT21] Tenindra Abeywickrama, Victor Liang, and Kian-Lee Tan. Optimizing bipartite matching in real-world applications by incremental cost computation. *Proceedings of the VLDB Endowment*, 14(7):1150–1158, 2021.

[ANFSW19] Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and Euclidean k-median by primal-dual algorithms. *SIAM Journal on Computing*, 49(4):FOCS17–97–FOCS17–156, 2019.

[ANOY14] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 574–583, 2014.

[ANWR17] Jason Altschuler, Jonathan Niles-Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. *Advances in neural information processing systems*, 30, 2017.

[AS14] Pankaj K Agarwal and R Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 555–564, 2014.

[AV07] David Arthur and Sergei Vassilvitskii. K-means++ the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007.

[AV09] David Arthur and Sergei Vassilvitskii. Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method. *SIAM J. Comput.*, 39(2):766–782, 2009.

[AZ23] Alexandr Andoni and Hengjie Zhang. Sub-quadratic (1+\eps)-approximate euclidean spanners, with applications. *arXiv preprint arXiv:2310.05315*, 2023.

[BBC$^+$19] Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for k-means: beyond subspaces and the johnson-lindenstrauss lemma. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1039–1050, 2019.

[BBK$^+$23] Ioana O Bercea, Lorenzo Beretta, Jonas Klausen, Jakob Bæk Tejs Houen, and Mikkel Thorup. Locally uniform hashing. *To appear at FOCS 2023, arXiv preprint arXiv:2308.14134*, 2023.

[BCALP23] Lorenzo Beretta, Vincent Cohen-Addad, Silvio Lattanzi, and Nikos Parotsidis. Multi-swap k-means++. *To appear at NeurIPS 2023, arXiv preprint arXiv:2309.16384*, 2023.

[BCIS05] Mihai Bădoiu, Artur Czumaj, Piotr Indyk, and Christian Sohler. Facility location in sublinear time. In *Automata, Languages and Programming: 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005. Proceedings 32*, pages 866–877. Springer, 2005.

[BJKS18] Jose Blanchet, Arun Jambulapati, Carson Kent, and Aaron Sidford. Towards optimal running times for optimal transport. *arXiv preprint arXiv:1810.07717*, 2018.

[BKS23] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1+\epsilon)$-approximate matching size in truly sublinear update time. *arXiv preprint arXiv:2302.05030*, 2023.

[BR23] Lorenzo Beretta and Aviad Rubinstein. Approximate earth mover's distance in truly-subquadratic time. *arXiv preprint arXiv:2310.19514*, 2023.

[BS83] Brenda S. Baker and Jerald S. Schwarz. Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing*, 12(3):508–525, 1983.

[BT22] Lorenzo Beretta and Jakub Tětek. Better sum estimation via weighted sampling. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2303–2338. SIAM, 2022.

[CASS21] Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. Improved coresets and sublinear algorithms for power means in euclidean spaces. *Advances in Neural Information Processing Systems*, 34:21085–21098, 2021.

[CEMN22] Vincent Cohen-Addad, Hossein Esfandiari, Vahab S. Mirrokni, and Shyam Narayanan. Improved approximations for euclidean k-means and k-median, via nested quasi-independent sets. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1621–1628. ACM, 2022.

[CG19] Fan Chung and Ron Graham. Efficient packings of unit squares in a large square. *Discrete & Computational Geometry*, 2019.

[CGPR20] Davin Choo, Christoph Grunau, Julian Portmann, and Vaclav Rozhon. k-means++: few more steps yield constant approximation. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1909–1917. PMLR, 2020.

[Cha02] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[CJLW22] Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. New streaming algorithms for high dimensional emd and mst. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 222–233, 2022.

[CK19] Vincent Cohen-Addad and Karthik C. S. Inapproximability of clustering in lp metrics. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 519–539. IEEE Computer Society, 2019.

[CKL21] Vincent Cohen-Addad, Karthik C. S., and Euiwoong Lee. On approximability of clustering problems without candidate centers. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2635–2648. SIAM, 2021.

[CKL$^+$22] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022.

[CKPT17] Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

[CLK22] Vincent Cohen-Addad, Euiwoong Lee, and Karthik C. S. Johnson coverage hypothesis: Inapproximability of k-means and k-median in ℓ_p metrics. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*. SIAM, 2022.

[CLSS22] Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, and Chris Schwiegelshohn. Towards optimal lower bounds for k-median and k-means coresets. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1038–1051. ACM, 2022.

[CR14] Clément Canonne and Ronitt Rubinfeld. Testing probability distributions underlying aggregated data. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 283–295, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[CS09] Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(3):904–922, 2009.

[Cut13] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

[CW97] János Csirik and Gerhard J. Woeginger. Shelf algorithms for on-line strip packing. *Information Processing Letters*, 63(4):171–175, 1997.

[CW98] János Csirik and Gerhard J. Woeginger. On-line packing and covering problems. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 147–177. Springer, 1998.

[DGK18] Pavel Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. Computational optimal transport: Complexity by accelerated gradient descent is better than by sinkhorn's algorithm. In *International conference on machine learning*, pages 1367–1376. PMLR, 2018.

[DKRT14] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. The power of two choices with simple tabulation. *CoRR*, abs/1407.6846, 2014.

[DKRT15] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. Hashing for statistics over k-partitions. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1292–1310. IEEE, 2015.

[DKS14] Anirban Dasgupta, Ravi Kumar, and Tamas Sarlos. On estimating the average degree. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, page 795–806, New York, NY, USA, 2014. Association for Computing Machinery.

[DM97] Karen L. Daniels and Victor Milenkovic. Multiple translational containment. part I: an approximate algorithm. *Algorithmica*, 19(1/2):148–182, 1997.

[DT14] Søren Dahlgaard and Mikkel Thorup. Approximately minwise independence with twisted tabulation. In *Proc. 14th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 134–145, 2014.

[EG75] Paul Erdős and Ron Graham. On packing squares with equal squares. *Journal of Combinatorial Theory, Series A*, 19(1):119–123, 1975.

[ERS17] Talya Eden, Dana Ron, and C. Seshadhri. Sublinear time estimation of degree distribution moments: The degeneracy connection. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 80. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, jul 2017.

[EvS18] Leah Epstein and Rob van Stee. Multidimensional packing problems. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, pages 553–570. Chapman and Hall/CRC, second edition, 2018.

[Fei06] Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM J. Comput.*, 35:964–984, 01 2006.

[FFGM07] Philippe Flajolet, Eric Fusy, Olivier Gandouet, and Frededric Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *In Analysis of Algorithms (AOFA)*, 2007.

[FH17a] Sándor P. Fekete and Hella-Franziska Hoffmann. Online square-into-square packing. *Algorithmica*, 77(3):867–901, 2017.

[FH17b] Sándor P. Fekete and Hella-Franziska Hoffmann. Online square-into-square packing. *Algorithmica*, 77(3):867–901, 2017.

[FW98] Amos Fiat and Gerhard J. Woeginger. Competitive analysis of algorithms. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 1–12. 1998.

[GI03] Venkatesan Guruswami and Piotr Indyk. Embeddings and non-approximability of geometric problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 537–538, 2003.

[GOR$^+$22] Fabrizio Grandoni, Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Rakesh Venkat. A refined approximation for euclidean k-means. *Inf. Process. Lett.*, 176:106251, 2022.

[GR06] Oded Goldreich and Dana Ron. Approximating average parameters of graphs. In Josep Díaz, Klaus Jansen, José D. P. Rolim, and Uri Zwick, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 363–374, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[HCT$^+$11] Xin Han, Francis Y. L. Chin, Hing-Fung Ting, Guochuan Zhang, and Yong Zhang. A new upper bound 2.5545 on 2d online bin packing. *ACM Transactions on Algorithms*, 7(4):50:1–50:18, 2011.

[HIYZ07] Xin Han, Kazuo Iwama, Deshi Ye, and Guochuan Zhang. Strip packing vs. bin packing. In *Algorithmic Aspects in Information and Management (AAIM 2007)*, pages 358–367, 2007.

[HPIS13] Sariel Har-Peled, Piotr Indyk, and Anastasios Sidiropoulos. Euclidean spanners in high dimensions. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 804–809. SIAM, 2013.

[HT52] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952.

[HT01] Eva Hopper and Brian Turton. A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artificial Intelligence Review*, 16(4):257–300, 2001.

[HT22] Jakob Bæk Tejs Houen and Mikkel Thorup. Understanding the moments of tabulation hashing via chaoses. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 74:1–74:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[Ind03] Piotr Indyk. Fast color image retrieval via embeddings. In *Workshop on Statistical and Computational Theories of Vision (at ICCV), 2003*, 2003.

[JV01] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[KLS11] Liran Katzir, Edo Liberty, and Oren Somekh. Estimating sizes of social networks via biased sampling. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 597–606, New York, NY, USA, 2011. Association for Computing Machinery.

[KMN$^+$04] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2):89–112, 2004. Special Issue on the 18th Annual Symposium on Computational Geometry - SoCG2002.

[Knu63] Donald E. Knuth. Notes on open addressing. Unpublished memorandum. See `http://citeseer.ist.psu.edu/knuth63notes.html`, 1963.

[KSKW15] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966. PMLR, 2015.

[Kuh55] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[KW12] Toryn Qwyllyn Klassen and Philipp Woelfel. Independence of tabulation-based hash classes. In *Proc. 10th Latin American Theoretical Informatics (LATIN)*, pages 506–517, 2012.

[LG03] Boris D. Lubachevsky and Ronald L. Graham. Dense packings of congruent circles in rectangles with a variable aspect ratio. In Boris Aronov, Saugata Basu, János Pach, and Micha Sharir, editors, *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, pages 633–650. 2003.

[LG09] Boris D. Lubachevsky and Ronald L. Graham. Minimum perimeter rectangles that enclose congruent non-overlapping circles. *Discrete Mathematics*, 309(8):1947–1962, 2009.

[Llo57] SP Lloyd. Least square quantization in pcm. bell telephone laboratories paper. published in journal much later: Lloyd, sp: Least squares quantization in pcm. *IEEE Trans. Inform. Theor.(1957/1982)*, 18, 1957.

[LNN$^+$21] Khang Le, Huy Nguyen, Quang M Nguyen, Tung Pham, Hung Bui, and Nhat Ho. On robust optimal transport: Computational complexity and barycenter computation. *Advances in Neural Information Processing Systems*, 34:21947–21959, 2021.

[LOZ12] Ping Li, Art B. Owen, and Cun-Hui Zhang. One permutation hashing. In *Proc. 26thAdvances in Neural Information Processing Systems*, pages 3122–3130, 2012.

[LS19a] Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3662–3671. PMLR, 2019.

[LS19b] Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3662–3671. PMLR, 2019.

[LSW17] Euiwoong Lee, Melanie Schmidt, and John Wright. Improved and simplified inapproximability for k-means. *Inf. Process. Lett.*, 120:40–43, 2017.

[LW88] Hyun-Chan Lee and Tony C. Woo. Determining in linear time the minimum area convex hull of two polygons. *IIE Transactions*, 20(4):338–345, 1988.

[LXH23] Yiling Luo, Yiling Xie, and Xiaoming Huo. Improved rate of first order algorithms for entropic optimal transport. In *International Conference on Artificial Intelligence and Statistics*, pages 2723–2750. PMLR, 2023.

[MD99] Victor J. Milenkovic and Karen Daniels. Translational polygon containment and minimal enclosure using mathematical programming. *International Transactions in Operational Research*, 6(5):525–554, 1999.

[Mil96] Victor Milenkovic. Translational polygon containment and minimal enclosure using linear programming based restriction. In *28th Annual ACM Symposium on the Theory of Computing (STOC 1996)*, pages 109–118, 1996.

[Mil97] Victor Milenkovic. Multiple translational containment. part II: exact algorithms. *Algorithmica*, 19(1/2):183–218, 1997.

[Mil99] Victor J. Milenkovic. Rotational polygon containment and minimum enclosure using only robust 2D constructions. *Computational Geometry*, 13(1):3–19, 1999.

[MMR19] Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Performance of johnson-lindenstrauss transform for k-means and k-medians clustering. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1027–1038. ACM, 2019.

[MPX07] Rajeev Motwani, Rina Panigrahy, and Ying Xu. Estimating sum by weighted sampling. In *Proceedings of the 34th International Conference on Automata, Languages and Programming*, ICALP'07, page 53–64, Berlin, Heidelberg, 2007. Springer-Verlag.

[MV08] Michael Mitzenmacher and Salil P. Vadhan. Why simple hash functions work: exploiting the entropy in a data stream. In *Proc. 19th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 746–755, 2008.

[OS18] Krzysztof Onak and Xiaorui Sun. Probability-revealing samples. In *AISTATS*, 2018.

[PBAA16] Dongwoo Park, Sang Won Bae, Helmut Alt, and Hee-Kap Ahn. Bundling three convex polygons to minimize area or perimeter. *Computational Geometry*, 51:1–14, 2016.

[PC19] Gabriel Peyré and Marco Cuturi. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[PLH$^+$20] Khiem Pham, Khang Le, Nhat Ho, Tung Pham, and Hung Bui. On unbalanced optimal transport: An analysis of sinkhorn algorithm. In *International Conference on Machine Learning*, pages 7673–7682. PMLR, 2020.

[PT10] Mihai Pătraşcu and Mikkel Thorup. On the k-independence required by linear probing and minwise independence. In *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 715–726, 2010.

[PT12a] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation-based hashing. *Journal of the ACM*, 59(3):Article 14, 2012. Announced at STOC'11.

[PT12b] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation-based hashing. *Journal of the ACM*, 59(3):Article 14, 2012. Announced at STOC'11.

[PT13] Mihai Pătraşcu and Mikkel Thorup. Twisted tabulation hashing. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–228, 2013.

[PVG$^+$11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, November 2011.

[Roh19] Dhruv Rohatgi. Conditional hardness of earth mover distance. *arXiv preprint arXiv:1909.11068*, 2019.

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40:99–121, 2000.

[San15] Filippo Santambrogio. Optimal transport for applied mathematicians. *Birkäuser, NY*, 55(58-63):94, 2015.

[SDGP$^+$15] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (ToG)*, 34(4):1–11, 2015.

[Ses15] C. Seshadhri. A simpler sublinear algorithm for approximating the triangle count. *CoRR*, may 2015.

[Spe13] E. Specht. High density packings of equal circles in rectangles with variable aspect ratio. *Computers & Operations Research*, 40(1):58 –69, 2013.

[Tho13] Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *FOCS*, pages 90–99, 2013.

[TT21] Jakub Tětek and Mikkel Thorup. Sampling and counting edges via vertex accesses. *arXiv e-prints*, page arXiv:2107.03821, 2021.

[TZ04] Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proc. 15th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 615–624, 2004.

[TZ09] Mikkel Thorup and Yin Zhang. Tabulation based 5-universal hashing and linear probing. In *Proc. 12th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2009.

[TZ12] Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing*, 41(2):293–331, 2012. Announced at SODA'04 and ALENEX'10.

[V$^+$09] Cédric Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009.

[vS12] Rob van Stee. SIGACT news online algorithms column 20: the power of harmony. *SIGACT News*, 43(2):127–136, 2012.

[vS15] Rob van Stee. SIGACT news online algorithms column 26: Bin packing in multiple dimensions. *SIGACT News*, 46(2):105–112, 2015.

[YCC$^+$19] Mikhail Yurochkin, Sebastian Claici, Edward Chien, Farzaneh Mirzazadeh, and Justin M Solomon. Hierarchical optimal transport for document representation. *Advances in neural information processing systems*, 32, 2019.

[YHZ09] Deshi Ye, Xin Han, and Guochuan Zhang. A note on online strip packing. *Journal of Combinatorial Optimization*, 17(4):417–423, 2009.

Appendix A

Unpublished: Approximate Earth Mover's Distance in Truly-Subquadratic Time

Approximate Earth Mover's Distance in Truly-Subquadratic Time

Lorenzo Beretta[1] and Aviad Rubinstein[2]

[1]BARC, University of Copenhagen, lorenzo2beretta@gmail.com
[2]Stanford University, aviad@cs.stanford.edu

Abstract

We design an additive approximation scheme for estimating the cost of the min-weight bipartite matching problem: given a bipartite graph with non-negative edge costs and $\varepsilon > 0$, our algorithm estimates the cost of matching all but $O(\varepsilon)$-fraction of the vertices in truly subquadratic time $O(n^{2-\delta(\varepsilon)})$.

- Our algorithm has a natural interpretation for computing the Earth Mover's Distance (EMD), up to a ε-additive approximation. Notably, we make no assumptions about the underlying metric (more generally, the costs do not have to satisfy triangle inequality). Note that compared to the size of the instance (an arbitrary $n \times n$ cost matrix), our algorithm runs in *sublinear* time.
- Our algorithm can approximate a slightly more general problem: max-cardinality bipartite matching with a knapsack constraint, where the goal is to maximize the number of vertices that can be matched up to a total cost B.

1 Introduction

Earth Mover's Distance (EMD - sometimes also Optimal Transport, Wasserstein-1 Distance or Kantorovich–Rubinstein Distance) is perhaps the most important and natural measure of similarity between probability distributions over elements of a metric space [PC19; San15; Vil+09]. Formally, given two probability distributions μ and ν over a metric space $(\mathcal{M}, d)$ their EMD is defined as

$$\mathrm{EMD}(\mu,\nu) = \min\left\{ \mathrm{E}_{(x,y)\sim\zeta}[d(x,y)] \;\middle|\; \zeta \text{ is a coupling}^1 \text{of } \mu \text{ and } \nu \right\}. \tag{1}$$

When μ and ν are discrete distributions with support size n (perhaps after a discretization preprocessing), a straightforward algorithm for estimating their EMD is to sample $\Theta(n)$ elements from each, compute all $\Theta(n^2)$ pairwise distances, and then compute a bipartite min-weight perfect matching. This algorithm clearly takes at least $\Omega(n^2)$ time (even ignoring the computation of the matching), and incurs a small additive error due to the sampling.

Our main result is an asymptotically faster algorithm for estimating the EMD:

[1]A distribution ζ over $\mathcal{M}^2$ is a coupling of μ and ν if $\mu(x) = \int_{\mathcal{M}} \zeta(x,y)\,dy$ and $\nu(y) = \int_{\mathcal{M}} \zeta(x,y)\,dx$.

Theorem 1 (Main Theorem)**.** *Suppose we have sample access to two distributions μ, ν over metric space $(\mathcal{M}, d)$ satisfying $d(\cdot,\cdot) \in [0,1]$ and query access to d. Suppose further that μ, ν have support size at most n.*

For each constant $\gamma > 0$ there exists a constant $\varepsilon > 0$ and an algorithm running in time $O(n^{2-\varepsilon})$ that outputs $\widehat{\mathrm{EMD}}$ such that

$$\widehat{\mathrm{EMD}} \in [\mathrm{EMD}(\mu,\nu) \pm \gamma].$$

Moreover, such algorithm takes $\tilde{O}(n)$ samples from μ and ν.

Notably, our algorithm makes no assumption about the structure of the underlying metric. In fact, it can be an arbitrary non-negative cost function, i.e. we do not even assume triangle inequality.

Beyond bounded support size. Support size is a brittle matter; indeed two distributions that are arbitrarily close in total variation (TV) distance (or EMD) can have completely different support size. Moreover, for continuous distributions, the notion of support size is clearly inappropriate and yet we would like to compute their EMD through sampling. To obviate this issue, Corollary 1.1 generalize Theorem 1 to distributions that are *close* in EMD to some distributions with support size n.

Corollary 1.1. *Suppose we have sample access to two distributions μ, ν over metric space $(\mathcal{M}, d_{\mathcal{M}})$ satisfying $d(\cdot,\cdot) \in [0,1]$ and query access to d. Suppose further that there exist μ', ν' with support size n such that $\mathrm{EMD}(\mu,\mu'), \mathrm{EMD}(\nu,\nu') \leqslant \xi$, for some $\xi > 0$.*

For each constant $\gamma > 0$ there exists a constant $\varepsilon > 0$ and an algorithm running in time $O(n^{2-\varepsilon})$ that outputs $\widehat{\mathrm{EMD}}$ such that

$$\widehat{\mathrm{EMD}} \in [\mathrm{EMD}(\mu,\nu) \pm (4\xi + \gamma)].$$

Moreover, such algorithm takes $\tilde{O}(n)$ samples from μ and ν.

For continuous μ, requiring that μ is close in EMD to a distribution with bounded support size is equivalent to saying that μ can be discretized effectively for EMD computation. Thus, such assumption is natural while computing EMD between continuous distribution through discretization.

We stress that the algorithm in Corollary 1.1 does not assume knowledge of μ' (nor ν') beyond its support size n. Indeed, the empirical distribution over $\tilde{O}(n)$ samples from μ (resp. ν) makes a good approximation in EMD. Finally, the sample complexity in Theorem 1 and Corollary 1.1 is optimal, up to polylog(n) factors. Indeed, Theorem 1 in [VV10] implies a lower bound of $\tilde{\Omega}(n)$ on the sample complexity of testing EMD closeness[2].

Matching with knapsack constraint. Applying our main algorithm to a graph-theory setting, we give an approximation scheme for a knapsack bipartite matching problem, where our goal is to estimate the number of vertices that can be matched subject to a total budget constraint.

Theorem 2 (Main theorem, graph interpretation)**.** *For each constant $\gamma > 0$, there exists a constant $\varepsilon > 0$, and an algorithm running in time $O(n^{2-\varepsilon})$ with the following guarantees. The algorithm takes as input a budget B, and query access to the edge-cost matrix of an undirected, bipartite graph G over n vertices. The algorithm returns an estimate $\widehat{M}$ that is within $\pm\gamma n$ of the size of the maximum matching in G with total cost at most B.*

[2]While deriving the lower bound from [VV10] takes some work, Remark 5.13 in [Can20] explicitly states a $\Omega(n/\log n)$ lower bound for TV closeness testing.

1.1 Relaed Work

Computing EMD is an important problem in machine learning [PC19] with some exemplary applications in computer vision [ACB17; RTG00; Sol+15] and natural language processing [Kus+15; Yur+19]. See [PC19] for a comprehensive overview.

Exact solution. Computing EMD between two sets of n points boils down to computing the minimum cost of a perfect matching on a bipartite graph, a problem with a 70-years history [Kuh55]. Min-weight bipartite perfect matching can be cast as a min-cost flow (MCF) instance and to date we can solve it in $n^{2+o(1)}$ time (namely, near-linear in the size of the distance matrix) [Che+22a]. Apparently, any exact algorithm requires inspecting the entire distance matrix, thus $\Theta(n^2)$ time is the best we can hope for. In addition, even in d-dimensional Euclidean space, where the input has size $d \cdot n \ll n^2$, no $O(n^{2-\varepsilon})$ algorithm exists[3], unless SETH is false [Roh19].

Multiplicative approximation. A significant body of work has investigated multiplicative approximation of EMD [Aga+22; AIK08; And+09; And+14; AS14; AZ23; Cha02; Che+22b; Ind03], where the most commonly studied setting is the Euclidean space (or, more generally, ℓ_p). If the dimension is constant we have near-linear time approximation schemes [Aga+22; And+14; FL22; SA12], whereas the high-dimensional case is more challenging. Only recently [AZ23] broke the $O(n^2)$ barrier for $(1+\varepsilon)$-approximation of EMD, building on [HIS13].

The landscape is much less interesting for general metrics. Indeed, a straightforward counterexample from [Bǎd+05] shows that any $O(1)$-approximation requires $\Omega(n^2)$ queries to the distance matrix. This suggests that for general metrics we should content ourselves with a additive approximation.

Additive approximation. Additive approximation for EMD has been extensively studied by optimization and machine learning communities [ANR17; Bla+18; Cut13; DGK18; Le+21; LXH23; Pha+20].

An extremely popular algorithm to solve optimal transport in practice is Sinkhorn algorithm [Cut13] (see [Le+21; Pha+20] for recent work). Sinkhorn distance SNK is defined by adding an entropy regularization term $-\eta \cdot H(\zeta)$ to the EMD objective in Equation (1). Approximating SNK via Sinkhorn algorithm provably yields a εr-additive approximation to EMD and takes $O_\varepsilon(n^2)$ time, where r is the dataset diameter [ANR17].

Graph-theoretic approaches also led to εr-additive approximations [LMR19] in $O_\varepsilon(n^2)$ time. Notice that even though all previous approximation algorithms have roughly the same complexity as the MCF-based exact solution they are backed by experiments showing their practicality, whereas exact algorithms for EMD are still largely impractical for very large graphs.

Breaking the $O(n^2)$ barrier for general metrics. As mentioned above, [AZ23] was the first work to break the quadratic barrier for approximate EMD. Indeed, they show a $(1+\varepsilon)$-multiplicative approximation algorithm for EMD on Euclidean space running in $n^{2-\Omega_\varepsilon(1)}$ time. Matching such result on general metrics is impossible, since no $O(1)$-multiplicative approximation can be achieved in $o(n^2)$ time [Bǎd+05]. A natural way to bypass the lower bound in [Bǎd+05] is to consider additive approximation. However, no ε-additive approximation algorithm for EMD on general

[3]The lower bound in [Roh19] holds in dimension $d = 2^{\Omega(\log^* n)}$.

metrics faster than $O_\varepsilon(n^2)$ barrier was known prior to this work. Theorem 1 gives the first ε-additive approximation to EMD for general metrics running in $n^{2-\Omega_\varepsilon(1)}$ time, thus breaking the quadratic barrier for general metrics.

We stress that, despite [AZ23] and this work both prove similar results, they use a completely different set of techniques. Indeed, in [AZ23] they approximate the complete bipartite weighted graph induced by Euclidean distances with a $(1+\varepsilon)$-multiplicative spanner of size $n^{2-\Omega_\varepsilon(1)}$. Their spanner construction is based on LSH and so it hinges on the Euclidean structure. Then, they run a near-linear time MCF solver [Che+22a] to solve the matching problem on the metric induced by the spanner. In this work, instead, we build on sublinear algorithms for max-cardinality matching [Beh+23; Beh22; BKS23a; BKS23b; BRR23] and do not leverage any metric property, not even triangle inequality. Section 2 contains a detailed explanation of our techniques.

It is worth to notice that since [AZ23] operates over d-dimensional Euclidean space the input representation takes $d \cdot n$ space, and so it *does not* run in sublinear time. On the contrary, our algorithm assumes query access to the distance matrix and runs in sublinear time.

Sublinear algorithms. Most previous work in sublinear models of computation focuses on streaming Euclidean EMD [AIK08; And+09; Bac+20; Cha02; Che+22b; Ind04], where the latest work [Che+22b] achieves $\tilde{O}(\log n)$-approximation in polylogarithmic space. Some other work [Ba+11] addresses the sample complexity of testing EMD on low-dimensional ℓ_1.

In this work we focus on a different access model: we do not make any assumption on the ground metric and we assume query access to the distance matrix. This model is natural whenever the underlying metric is expensive to evaluate. For example, in [ALT21] they consider EMD over a shortest-path ground metric and experiment with heuristics to avoid computing all-pair distances, which would be prohibitively expensive.

Comparison with MST. Minimum Spanning Tree (MST) and EMD are two of the most studied optimization problems in metric spaces. It is interesting to observe a separation between the sublinear-time complexity of MST and EMD for general metrics. Indeed, [CS09] shows a $\tilde{O}_\varepsilon(n)$ time algorithm approximating the *cost* of MST up to a factor $1+\varepsilon$, whereas no $O(1)$-approximation for EMD can be computed in $o(n^2)$ time [Bǎd+05]. Essentially, this is due to the fact that MST cost is a more *robust* problem than EMD. Indeed, in EMD increasing a single entry in the distance matrix can increase the EMD arbitrarily, whereas for MST this does not happen because of triangle inequality.

A valuable take-home message from this work is that allowing additive approximation makes EMD more robust. A natural question is whether we can find a ε-additive approximation to EMD in $\tilde{O}_\varepsilon(n)$ time, thus matching the above result on MST cost. The $\Omega(n^{1.2})$ lower bound on max-cardinality matching from [BRR23] suggests that this should not be possible[4] Indeed, we can reduce max-cardinality matching to EMD by embedding the bipartite graph into a $(1, 2)$ metric space.

2 Technical Overview

Computing Earth Mover's Distance between two sets of n points in a metric space can be achieved by solving Min-Weight Perfect Matching (MWPM) on the complete bipartite graph where edge-

[4]The lower bound of [BRR23] is proven in a slightly different model of adjacency list.

costs are given by the metric $d(\cdot,\cdot)$. Here we seek a suitable notion of approximation for MWPM that recovers Theorem 1.

Min-weight perfect matching with outliers. Consider the following problem: given a constant $\gamma > 0$, find a matching M of size $(1-\gamma)n$ in a bipartite graph such that the cost of M is at most the minimum cost of a perfect matching. A natural interpretation of this problem is to label a γ fraction of vertices as outliers and leave them unmatched; so we dub this problem MWPM *with outliers.*

Assuming $d(\cdot,\cdot) \in [0,1]$, solving MWPM with a γ fraction of outliers immediately yields a γ additive approximation to EMD, proving Theorem 1.

The main technical contribution of this work is the following theorem, which introduces an algorithm that solves MWPM with outliers in sublinear time. For the sake of this overview, the reader should instantiate Theorem 3 with $\beta = 1$ and think of $\gamma = (1-\alpha)$ as the fraction of allowed outliers.

Theorem 3. *For each constants $0 \leqslant \alpha < \beta \leqslant 1$ there exists a constant $\varepsilon > 0$ and an algorithm running in time $O(n^{2-\varepsilon})$ with the following guarantees.*

The algorithm has adjacency-matrix access to an undirected, bipartite graph $G = (V_0 \cup V_1, E)$ and random access to the edge-cost function $c : E \to \mathbb{R}^+$. The algorithm returns $\hat{c}$ such that, whp,

$$c(M^\alpha) \leqslant \hat{c} \leqslant c(M^\beta)$$

where M^α is a minimum-weight matching of size αn and M^β is a minimum-weight matching of size βn.

Moreover, the algorithm returns a matching oracle data structure that, given a vertex u returns, in $n^{1+f(\varepsilon)}$ time, an edge $(u,v) \in \hat{M}$ or $\perp$ if $u \notin V(\hat{M})$, where $f(\varepsilon) \to 0$ when $\varepsilon \to 0$. The matching $\hat{M}$ satisfies $\alpha n \leqslant |\hat{M}| \leqslant \beta n$ and $c(M^\alpha) \leqslant c(\hat{M}) \leqslant c(M^\beta)$.

Notice that the algorithm in Theorem 3 does *not* output the matching $\hat{M}$ explicitly. However, it returns a matching oracle data structure which implicitly stores $\hat{M}$. The rest of this overview sketches the proof of Theorem 3.

Our algorithm, in a nutshell. A new set of powerful techniques was recently developed to approximate the size of a max-cardinality matching in sublinear time [Beh+23; Beh22; BKS23a; BKS23b; BRR23]. Our main contribution is a sublinear-time algorithm which leverages the techniques above to implement (a certain step of) the classic Gabow-Tarjan [GT89] algorithm for MWPM. Since the techniques above return *approximate* solutions, the obtained matching will be approximate as well, in the sense that we have to disregard a fraction of outliers when computing its cost to recover a meaningful guarantee. Careful thought is required for relaxing the definitions of certain objects in the Gabow-Tarjan algorithm so as to accommodate their computation in sublinear time. The bulk of our analysis is devoted to proving that these relaxations combine well and lead to the guarantee in Theorem 3.

Roadmap. First, we will review (a certain step of) the Gabow-Tarjan algorithm that we will use as our template algorithm to be implemented in sublinear time. Then, we will review some recent sublinear algorithms for max-cardinality matching. Finally, we will sketch how to combine these tools to approximate the value of minimum-weight matching.

2.1 A Template Algorithm

The original Gabow-Tarjan algorithm operates on several scales and this makes it (slightly) more involved. We focus here on a simpler case where all our edge weights are integers in $[1, C]$, for $C = O(1)$. We will see in Section 6 that we can reduce to this case (incurring a small additive error). Here we describe our template algorithm, at a high level.

A linear program for MPWM. First, recall the linear program for MWPM together with its dual. Here we consider a bipartite graph $G(V = V_0 \cup V_1, E)$ and cost function $c(\cdot,\cdot) \in [1, C]$. We can interpret the following LP so that $x_{u,v} = 1$ iff u and v are matched, whereas primal constraints require every vertex to be matched.

Primal

$$\begin{aligned} \text{Minimize} \quad & \sum_{u\in V_0, v\in V_0} x_{u,v} \cdot c(u,v) \\ \text{subject to} \quad & \sum_{v\in V_1} x_{u,v} \geqslant 1 \quad \forall u \in V_0 \\ & \sum_{u\in V_0} x_{u,v} \geqslant 1 \quad \forall v \in V_1 \\ & x_{u,v} \geqslant 0 \quad \forall u \in V_0,\ v \in V_1. \end{aligned}$$

Dual

$$\begin{aligned} \text{Maximize} \quad & \sum_{u\in V} \varphi_u \\ \text{subject to} \quad & \varphi_u + \varphi_v \leqslant c(u,v) \quad \forall (u,v) \in E \\ & \varphi_u \geqslant 0. \quad \forall u \in V, \end{aligned}$$

A high-level description. Essentially, our template algorithm is a primal-dual algorithm which (implicitly) maintains a pair (M, φ), where M is a partial matching (so primal infeasible), and $\{\varphi(v)\}_{v\in V}$ is a vertex potential function, or an (approximately) feasible dual solution. Moreover, for each $e \in M$ the dual constraint corresponding to e is tight. In other words, the pair (M, φ) satisfies complementary slackness. The algorithm progressively grows the dual variables $\{\varphi(v)\}_{v\in V}$ and the size of M. When M has size $\geqslant (1-\gamma)n$ then we are done. Indeed, throwing out γn vertices (as well as their associated primal constraints) we have that (M, φ) is a (approximately) feasible primal-dual pair that satisfies complementary slackness, thus it is (approximately) optimal.

The primal-dual algorithm. We maintain an initially empty matching M. Inspired by the dual, we define a potential function $\varphi : V \to \mathbb{Z}$ and we enforce a relaxed version of the dual constraints: $\varphi(u) + \varphi(v) \leqslant c(u,v) + 1$ for each $(u,v) \in E$. Moreover, we maintain that $\varphi(u) + \varphi(v) = c(u,v)$ for each $(u,v) \in M$ (complementary slackness). Let T be the set of edges s.t. the constraints above are tight. Orient the edges in T so that all edges in $M \subseteq T$ are oriented from V_0 to V_1 and all edges in $T \backslash M$ are oriented from V_1 to V_0. We denote the set of free (unmatched) vertices F and let $F_0 = F \cap V_0$ $F_1 = F \cap V_1$. We say that a path $P = (v_0 \to \cdots \to v_1)$ is an augmenting path if $v_0 \in F_0$, $v_1 \in F_1$ and P alternates between edges in $T \backslash M$ and M. When we say that we augment M wrt P we mean that we set $M \leftarrow M \oplus P$. We alternate between the following two steps:

1. Find a a large set of node-disjoint augmenting paths $\{P_1 \dots P_\ell\}$. Augment M wrt these paths. Decrement $\varphi(v)$ `-=` 1 for each $v \in \bigcup_i P_i \cap V_1$, to ensure the relaxed dual constraints are satisfied.

2. Define R as the set of vertices that are T-reachable[5] from F_0. Increment $\varphi(r_0)$ `+=` 1 for each $r_0 \in R \cap V_0$, and decrement $\varphi(r_1)$ `-=` 1 for each $r_1 \in R \cap V_1$. This preserves the relaxed dual constraints and (eventually) adds some more edges to T.

After $O_{\gamma,C}(1)$ iterations, we have $|F| \leqslant \gamma n$.

Analysis sketch. It is routine to verify that steps 1 and 2 preserve the relaxed dual constraints. At any point the pair (M, φ) satisfies, $c(M) \leqslant \sum_{v \in V_0 \cup V_1} \varphi(v) \leqslant c(M') + n$ for any perfect matching M'. We can content ourselves with this additive approximation; indeed in Section 6 we will see how to charge it on the outliers. To argue that we have few free vertices left after $O_\gamma(1)$ iterations, notice that at iteration t we have $\varphi|_{F_0} \equiv t$ and $\varphi|_{F_1} \equiv 0$. Computing a certain function of potentials along $(M \oplus M')$-augmenting paths shows that $|F| \cdot t \leqslant O(n)$. Thus, $O_\gamma(1)$ iterations are sufficient to obtain $|F| \leqslant \gamma n$. The arguments above are sufficient to show that our template algorithm finds an (almost) perfect matching with (almost) minimum weight. We will shove both *almost* under the outlier carpet in Section 6.

2.2 Implementing the Template in Sublinear Time

Our sublinear-time implementation of the template algorithm hinges on matching oracles.

Matching oracles. Given a matching M' we define a *matching oracle* for M' as a data structure that given $u \in V$ returns $v \in V$ if $(u, v) \in M'$ and $\perp$ otherwise. Note that given a matching oracle for M', if we are promised that $|M'| = \Omega(n)$ then $O_\gamma(\log n)$ calls to such oracle are enough to estimate $|M'| \pm \gamma n$. We stress that all matching oracles that we use have sublinear query time.

Finding large matchings in sublinear time. An important ingredient in our algorithm is the `LargeMatching`$(G, A, \varepsilon, \delta)$ subroutine (Theorem 5), which is due to [BKS23a]. Given $A \subseteq V$, `LargeMatching`$(G, A, \varepsilon, \delta)$ returns either $\perp$ or a matching oracle for some matching M' in $G[A]$. If there exists a matching in $G[A]$ of size δn, then `LargeMatching` returns a matching oracle for some M' in $G[A]$ with $|M'| = \Omega_\delta(n)$. Else, if there are no matchings of size δn in $G[A]$ `LargeMatching` returns $\perp$. The parameter ε controls the running time and essentially guarantees that `LargeMatching` runs in $O(n^{2-\varepsilon})$ time while the matching oracle it outputs runs in $O(n^{1+\varepsilon})$.

We will use `LargeMatching` to implement both step 1 and step 2 in the template algorithm. However, this requires us to relax our notions of maximal set of node-disjoint augmenting paths, as well as that of reachability. A major technical contribution of this work is to find the right relaxation of these notions so that:

1) We can analyze a variant of the template algorithm working with these relaxed objects and still recover a a solution which is optimal if we neglect a γ fraction of outliers.

2) We can compute these relaxed objects in sublinear time using `LargeMatching` as well as previously constructed matching oracles.

These relaxed notions are introduced in Section 3, point (1) is proven in Section 4 and point (2) is proven in Section 5.

[5]Recall that T is oriented.

Implementing step 1 in sublinear time. In [BKS23a] the authors implement McGregor's algorithm [McG05] for streaming Max-Cardinality Matching (MCM) in a sublinear fashion using `LargeMatching` (see Theorem 6 in this work). McGregor's algorithm finds a size-$\Omega(n)$ set of node-disjoint augmenting paths of fixed constant length, whenever there are at least $\Omega(n)$ of them. This notion is weaker than that of a maximal node-disjoint set of augmenting paths required in step 1 of our template algorithm in two regards: first, it only finds augmenting paths of fixed constant length; second, it finds only a constant fraction of such paths (as long as we have a linear number of them).

In our template algorithm, the invariant $\varphi|_{F_1} \equiv 0$ is maintained (in step 2) because $R \cap F_1 = \varnothing$. In turn, $R \cap F_1 = \varnothing$ holds exactly because in step 1 we augment M with a maximal node-disjoint set of augmenting paths. Since our sublinear implementation of step 1 misses some augmenting paths, the updates performed in step 2 will violate the invariant $\varphi(v) = 0$ for some $v \in F_1$.

A careful implementation of step 2 (see next paragraph) guarantees that only missed augmenting paths that are short lead to a violation of $\varphi|_{F_1} \equiv 0$. Moreover, repeatedly running the sublinear implementation of McGregor's algorithm from [BKS23a], we ensure that we miss at most γn short paths, for γ arbitrary small. Thus, we can flag all vertices that belong to missed short augmenting paths as outliers since we have only a small fraction of them.

Implementing step 2 in sublinear time. We implement an approximate version of the reachability query in step 2 as follows. We initialize the set of reachable vertices R as $R \leftarrow F_0$. Then, for a constant number of iterations: we compute a large matching $M' \subseteq T \backslash M$ between the vertices of $R \cap V_0$ and $V_1 \backslash R$; then we add to R all matched vertices in $\bigcup M'$ as well as their M-mates, namely $\texttt{mate}_M(u)$ for each $u \in \bigcup M'$. Notice that if a $\Omega(n)$-size matching $\subseteq T \backslash M$ between $R \cap V_0$ and $V_1 \backslash R$ exists, then we find a matching $\subseteq T \backslash M$ between $R \cap V_0$ and $V_1 \backslash R$ of size at least $\Omega(n)$. This ensures that: (i) after a constant number of iterations `LargeMatching` returns $\perp$; (ii) when `LargeMatching` returns $\perp$ there exists a vertex cover $\mathcal{C}$ of $((R \cap V_0) \times (V_1 \backslash R)) \cap T \backslash M$ of size γn. Only constraints corresponding to edges incident to $\mathcal{C}$ might be violated during step 2. Furthermore, $|\mathcal{C}| = \gamma n$ is small and so we can just label vertices in $\mathcal{C}$ as outliers.

As we pointed out in the previous paragraph, the invariant $\varphi|_{F_1} \equiv 0$ might be violated in step 2 if $R \cap F_1 \neq \varnothing$. We already showed that whenever we the missed augmenting path causing the violation of $\varphi|_{F_1} \equiv 0$ is short we can charge this violation on a small set of outliers. To make sure that no long augmenting path leads to a violation of $\varphi|_{F_1} \equiv 0$ we set our parameters so that the depth of the reachability tree built in step 2 is smaller than the length of "long" paths. Thus, any long path escapes R and cannot cause a violation.

Everything is an oracle. The implementation of both step 1 and step 2 operates on the graph T of tight constraints. To evaluate $(u, v) \in T$, we need to compute $\varphi(u)$ and $\varphi(v)$. In turn, the potential values depend on previous iterations of the algorithm. None of these iterations outputs an explicit description of the objects described in the template (potentials, matchings, augmenting paths or sets of reachable vertices). Indeed, these objects are output as oracle data structures, which internals call (eventually multiple) matching oracles output by `LargeMatching`. We prove that essentially all these oracles have query time $O(n^{1+\varepsilon})$ for some small $\varepsilon > 0$. A careful analysis is required to show that we can build the oracles at iteration $i + 1$ using the oracles at iteration i without blowing up their complexity.

Paper organization. In Section 3 we define some fundamental objects that we will use throughout the paper. In Section 4 we present a template algorithm to be implemented in sublinear time, and prove its correctness. In Section 5 we implement the template algorithm in sublinear time. In Section 6 we put everything together and prove the main theorems stated in the introduction.

3 Preliminaries

We use the notation $[a, b] := \{a \dots b-1\}$, $[b] = [0, b]$, and $(a \pm b) := [a-b, a+b]$ meaning that $c \cdot (a \pm b) = (ac \pm bc)$. We denote our undirected bipartite graph with $G = (V, E)$, and the bipartition is given by $V = V_0 \cup V_1$. Our original graph is complete and for each $(u, v) \in V_0 \times V_1$ we denote with $c(u, v)$ the cost of the edge (u, v). We stress that none of our algorithms require $c(\cdot, \cdot)$ to be a metric. Given a matching M we denote its combined cost with $c(M)$. For each $u \in V$ we say that $u = \texttt{mate}_M(v)$ iff $(u, v) \in M$. When the matching M is clear from the context we denote with F the set of unmatched (or *free*) vertices, and set $F_i := F \cap V_i$ for $i = 0, 1$.

When we say that an algorithm runs in time t we mean that both its computational complexity and the number of queries to the cost matrix $c(\cdot, \cdot)$ are bounded by t. The computational complexity of our algorithms is always (asymptotically) equivalent to their query complexity, so we only analyse the latter. All our guarantees in this work hold with high probability.

Definition 3.1 (Augmenting paths)**.** *Given a matching M over $G = (V, E)$ we say that $P = (v_0, v_1 \dots v_{2\ell+1})$ is an augmenting path w.r.t. M if $(v_{2i}, v_{2i+1}) \in E \backslash M$ for each $i = 0 \dots \ell$ and $(v_{2j+1}, v_{2j+2}) \in M$ for each $j = 0 \dots \ell - 1$. When we say that we augment M w.r.t. P we mean that we set $M \leftarrow M \oplus P$, where $\oplus$ is the exclusive or.*

We use the same notion of 1-feasible potential as in [GT89].

Definition 3.2 (1-feasibility conditions)**.** *Given a potential $\varphi : V \longrightarrow \mathbb{Z}$ we say that it satisfies 1-feasibility conditions with respect to a matching M if the following hold.*

(i) For each $u \in V_0, v \in V_1$ $\varphi(u) + \varphi(v) \leqslant c(u, v) + 1$.

(ii) For each $(u, v) \in M$, $\varphi(u) + \varphi(v) = c(u, v)$.

Definition 3.3 (Eligibility Graph)**.** *We say that an edge (u, v) is eligible w.r.t. M if: $(u, v) \notin M$ and $\varphi(u) + \varphi(v) = c(u, v) + 1$ or; $(u, v) \in M$ and $\varphi(u) + \varphi(v) = c(u, v)$. We define the eligibility graph as the directed graph $G_{\mathcal{E}} = (V, E_{\mathcal{E}})$ that orients the eligible edges so that, for each eligible $(u, v) \in V_0 \times V_1$, we have $(u, v) \in E_{\mathcal{E}}$ if $(u, v) \notin M$ and $(v, u) \in E_{\mathcal{E}}$ if $(u, v) \in M$.*

Notice that, whenever a potential is 1-feasible w.r.t. M, then all edges in M are eligible.

Definition 3.4 (Forward Graph)**.** *We define the forward graph $G_F = (V, E_F)$ as the subgraph of the eligibility graph containing only edges from V_0 to V_1. That is, we remove all edges (v, u) such that $(u, v) \in M$.*

Now, we introduce two quite technical definitions, which provide us with approximate versions of the notion of "maximal set of node-disjoint augmenting paths" and "maximal forest".

Definition 3.5 ((k, ξ)-Quasi-Maximal Set of Node-Disjoint Augmenting Paths)**.** *Given a graph $G = (V, E)$ and a matching $M \subseteq E$ we say that a set $\mathcal{P}$ of augmenting paths of length at most k is a (k, ξ)-QMSNDAP if for any $\mathcal{Q}$ such that $\mathcal{Q} \cup \mathcal{P}$ is a set of node-disjoint augmenting paths of length $\leqslant k$ we have $|\mathcal{Q}| \leqslant \xi n$.*

Intuitively, $\mathcal{P}$ is a (k, ξ)-QMSNDAP if we can add only a few more node-disjoint augmenting paths of length $\leqslant k$ to $\mathcal{P}$ before it becomes a maximal. Next we introduce an approximate notion of "maximal forest" $\mathcal{F}$ in the eligibility graph $G_{\mathcal{E}}$ rooted in F_0. $\mathcal{F}$ is obtained starting from the vertices in F_0 and adding edges (in a way that we will specify later) so as to preserve the $\mathcal{F}$ has $|F_0|$ connected components and has no cycles. This construction will ensure that the connected component of our forest have small diameter and small size. We maintain that whenever $v \in V_1$ is added to $\mathcal{F}$, then $\mathtt{mate}_M(v)$ is also added to $\mathcal{F}$. $\mathcal{F}$ is approximately maximal in the sense that the cut $(\mathcal{F}, V \backslash \mathcal{F})$ in $G_{\mathcal{E}}$ admits a small vertex cover.

Definition 3.6 ((k, δ)-Quasi-Maximal Forest)**.** *Given the eligibility graph $G_{\mathcal{E}} = (V, E_{\mathcal{E}})$ w.r.t. the matching M, and the set of vertices $F_0 \subseteq V_0$ we say that $\mathcal{F}$ is a (k, δ)-QMF rooted in F_0 if:*

1. *$F_0 \subseteq \mathcal{F}$*
2. *For each $v \in V_1 \cap \mathcal{F}$ we have $\mathtt{mate}_M(v) \in \mathcal{F}$*
3. *For each $u \in \mathcal{F}$ there exists $v \in F_0$ at hop distance from u at most k.*
4. *Every connected component of $\mathcal{F}$ has size at most 2^k.*
5. *The edge set $E_{\mathcal{E}} \cap (\mathcal{F} \times V \backslash \mathcal{F})$ has a vertex cover of size at most δn.*

Now, we introduce a few results from past work on sublinear-time maximum caridnality matching. The following theorem, which is the main technical contribution of [BKS23a], states that we can compute a εn-additive approximation of the size of a maximum-cardinality matching in strongly sublinear time.

Theorem 4 (Theorem 1.3, [BKS23a])**.** *There is a randomized algorithm that, given the adjacency matrix of a graph G, in time $n^{2-\Omega_{\varepsilon}(1)}$ computes with high probability a $(1, \varepsilon n)$-approximation $\tilde{\mu}$ of $\mu(G)$. After that, given a vertex v, the algorithm returns in $n^{1+f(\varepsilon)}$ time an edge $(v, v') \in M$ or $\perp$ if $v \notin V(M)$ where M is a fixed $(1, \varepsilon n)$-approximate matching, where f is an increasing function such that $f(\varepsilon) \to 0$ when $\varepsilon \to 0$.*

The algorithm in Theorem 4 does not exactly output a matching, but rather a *matching oracle.* Namely, it outputs a data structure that stores a matching M implicitly. We formalize the notion of matching oracle below.

Definition 3.7 (Matching Oracle)**.** *Given a matching M, we define the matching oracle $\mathtt{match}_M(\cdot)$ as a data structure such that $\mathtt{match}_M(u) = v$ if $(u, v) \in M$ and $\mathtt{match}_M(u) = \perp$ otherwise. Throughout the paper we denote with t_M the time complexity of $\mathtt{match}_M(\cdot)$.*

Similarly to matching oracles, we make use of membership oracles $\mathtt{mem}_A(\cdot)$ and potential oracles $\mathtt{eval}_{\varphi}(\cdot)$ where $A \subseteq V$ and φ is a potential function defined on V. As expected, $\mathtt{mem}_A(u)$ returns $\mathbb{1}_{u \in A}$ and $\mathtt{eval}_{\varphi}(u)$ returns $\varphi(u)$. We denote their running time with t_A and t_{φ} respectively. Now, we recall two theorems from [BKS23a] that constitutes fundamental ingredients of our sublinear-time algorithm for minimum-weight matching.

Theorem 5 roughly says that, in sublinear time, we can find a matching oracle for a size-$\Omega(n)$ matching, whenever a size-$\Omega(n)$ matching exists.

Theorem 5 (Essentially Theorem 4.1, [BKS23a]). *Let $G = (V, E)$ be a graph, $A \subseteq V$ be a vertex set. Suppose that we have access to adjacency matrix of G and an A-membership oracle $\texttt{mem}_A$ with t_A query time. We are given as input a sufficiently small $\varepsilon > 0$ and $\delta_{\text{in}} > 0$.*

There exists an algorithm $\texttt{LargeMatching}(G, A, \varepsilon, \delta_{\text{in}})$ that preprocesses G in $\tilde{O}_{\delta_{\text{in}}}((t_A + n) \cdot n^{1-\varepsilon})$ time and either return $\perp$ or construct a matching oracle $\texttt{match}_M(\cdot)$ for a matching $M \subset G[A]$ of size at least $\delta_{\text{out}} n$ where $\delta_{\text{out}} = \frac{1}{2000}\delta_{\text{in}}^5$ that has $\tilde{O}_{\delta_{\text{in}}}((t_A + n)n^{4\varepsilon})$ worst-case query time. If $\mu(G[A]) \geqslant \delta_{\text{in}} n$, then $\perp$ is not returned. The guarantee holds with high probability.

Theorem 6 roughly says that, in sublinear time, we can increase the size of our current matching (oracle) by $\Omega(n)$, whenever there are $\Omega(n)$ short augmenting paths.

Theorem 6 (Essentially Theorem 5.2, [BKS23a]). *Fix two constants $k, \gamma > 0$. For any sufficiently small $\varepsilon_{in} > 0$, there exists $\varepsilon_{out} = \Theta_{k,\gamma}(\varepsilon_{in})$ such that the following holds. There exists an algorithm $\texttt{Augment}(G, M^{in}, k, \gamma, \varepsilon_{in})$ that makes $O_{k,\gamma}(1)$ calls to $\texttt{LargeMatching}$ which take $\tilde{O}_{k,\gamma}\left(n^{2-\varepsilon_{in}}\right)$ time in total. Further, either it returns an oracle $\texttt{match}_{M^{out}}(\cdot)$ with query time $\tilde{O}_{k,\gamma}(n^{1+\varepsilon_{out}})$, for some matching M^{out} in G of size $|M^{out}| \geqslant |M^{in}| + \Theta_{k,\gamma}(1) \cdot n$ (we say that it "succeeds" in this case), or it returns* FAILURE. *Finally, if the matching M^{in} admits a collection of $\gamma \cdot n$ many node-disjoint length $(2k+1)$-augmenting paths in G, then the algorithm succeeds whp.*

Theorem 6 differs from Theorem 5.2 in [BKS23a] in that it specifies that the only way `Augment` accesses the graph is through `LargeMatching`. We will use this property crucially to prove Lemma 5.2.

4 A Template Algorithm

In this section we study min-weight matching with integral small costs $c(\cdot, \cdot) \in [1, C]$, where C is constant. We will see how to lift this restriction in Section 6. Algorithm 1 gives a template algorithm realising Theorem 3 that assumes we can implement certain subroutines; in Section 5 we will see how to implement these subroutines in sublinear time.

Comparison with Gabow-Tarjan. Intuitively, our template algorithm implements the Gabow-Tarjan algorithm [GT89] for a fixed scale in an approximate fashion. Indeed, instead of finding a maximal-set of node-disjoint augmenting paths we find a (k, ξ)-QMSNDAP and instead of growing a forest in the eligibility graph we grow a (k, δ)-QMF. See Figure 1 for a representation of step 1 and step 2.

Analysis. Here we analyse Algorithm 1 and show that it satisfies the following theorem.

Theorem 7. *Fix a constant $\gamma > 0$. Suppose that we have adjacency-matrix access to the bipartite graph $G = (V_0 \cup V_1, E)$ and random access to the cost function $c : E \to [1, C]$, with $C = O(1)$. Then, with high probability, Algorithm 1 returns $\hat{c}$ such that*

$$c(M^{1-\gamma}) \leqslant \hat{c} \leqslant c(M^{OPT})$$

where $M^{1-\gamma}$ is a min-weight matching of size $(1-\gamma)n$ and M^{OPT} is a min-weight matching of size n.

To prove Theorem 7, we need a series of technical lemmas.

Algorithm 1 Template Algorithm.

```
Input: A bipartite graph $G = (V_0 \cup V_1, E)$ and a cost function $c : E \to [1, C]$.
Set $T = C/\gamma^3$, $\xi = \frac{\gamma}{Tk2^k}$, $\delta = \frac{\gamma}{T}$ and $k = 6000(2T+1)^{10}/\delta^5$.
Initialize $M \leftarrow \varnothing$ and $\varphi(v) \leftarrow 0$ for each $v \in V$.
Let $F_0$ denote the set of $M$-unmatched vertices in $V_0$.
For each $e \in E$ update $c(e) \leftarrow c(e)/\gamma$ (this is implemented lazily).
Execute the following two steps for $T$ iterations:
    • Step 1. Find a $(k, \xi)$-QMSNDAP $\mathcal{P}$ in the eligibility graph $G_{\mathcal{E}}$. Augment $M$ w.r.t. paths in
      $\mathcal{P}$. Set $\varphi(v) \leftarrow \varphi(v) - 1$ for each $v \in V_1 \cap \bigcup_{P \in \mathcal{P}} P$.
    • Step 2. Find a $(k, \delta)$-QMF$\mathcal{F}$ rooted in $F_0$ in the eligibility graph $G_{\mathcal{E}}$. Set $\varphi(u) \leftarrow \varphi(u) + 1$
      for each $u \in V_0 \cap \mathcal{F}$ and $\varphi(v) \leftarrow \varphi(v) - 1$ for each $v \in V_1 \cap \mathcal{F}$.
Sample a set $S$ of $O_{\gamma,C}(\log n)$ edges in $M$ with replacement.
Discard the $3\gamma|S|$ edges with highest costs and let $\Sigma$ be the sum of costs of remaining edges.
Output: $\hat{c} = \frac{n}{|S|}\Sigma$.
```

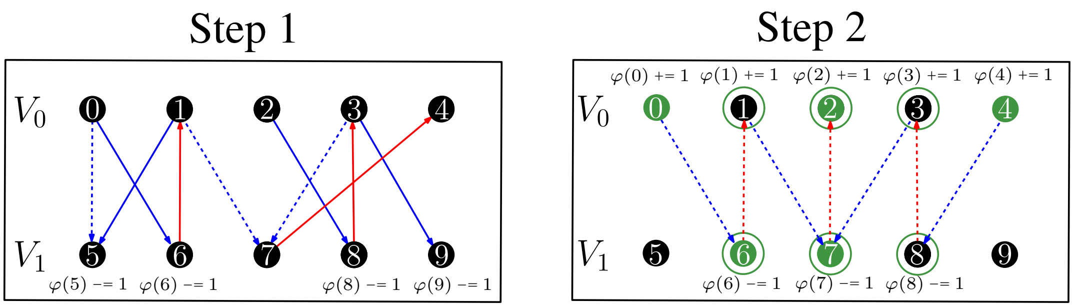


Figure 1: We color the edges of M in red and the edges of $T \backslash M$ in blue. On the left we have an example of step 1. Solid edges represent paths in the QMSNDAP $\mathcal{P}$ that we augment M along in step 1. On the right we have an example of step 2. All vertices colored or circled in green belong to the QMF $\mathcal{F}$. Circles help us visualize the implementation of step 2, described in Section 5. In Algorithm 3 $\mathcal{F}$ is built sequentially, where each iteration (lines 1-5) adds some edges to $\mathcal{F}$. At first, only the non-circled green vertices belong to $\mathcal{F}$. The first step adds the green-circled black edges, and the second step adds the green-circled green edges.

Proof Roadmap. The proof of Theorem 7 goes as follows. We prove that, after T iterations, all free vertices in F_0 have potential T. On the other hand, the majority of free vertices in F_1 have potential 0. We call *spurious* the free vertices in F_1 with non-zero potential and we show there are only few of them. Then, (roughly) we look at the final matching M generated by Algorithm 1 and a perfect matching M' and consider the graph G' having $M \oplus M'$ as its set of edges. G' can be partitioned into cycles and augmenting paths. Each augmenting path starts in a free vertex in F_0 and ends in a free vertex in F_1. If the 1-feasibility conditions are satisfied by all edges, then computing a certain function of potentials along an augmenting path and combining the results for all augmenting paths yields an upper bound on the total number of free vertices. Unfortunately, not all edges satisfy the 1-feasibility constraints. We fix this by finding a small vertex cover of the 1-unfeasible edges. We say that such cover a suitable set of *broken* vertices. Ignoring spurious and broken vertices is sufficient to make our argument work.

Lemma 4.1. *After $t \in [T+1]$ iterations we have $\varphi(u) = t$ for each $u \in F_0$.*

Proof. After $t = 0$ iterations, we have $\varphi(u) = 0$ for each $u \in V$. First, we notice that the set of unmatched (or free) vertices F only shrinks over time, and so does F_0. Moreover, at each iteration we increase the potential of free vertices in F_0 by 1. □

Define the set S of $v \in F_1$ such that $\varphi(v) \neq 0$ as the set of *spurious* vertices.

Lemma 4.2. *After T iterations we have at most γn* spurious *vertices.*

Proof. We prove that at each iteration we increase the number of spurious vertices by at most $\gamma n/T$. A vertex cannot become spurious in Step 1. Indeed, in Step 1 we only decrease the potential of matched vertices. If a vertex $v \in F_1$ becomes spurious in Step 2, it means that there exists an augmenting path P from some $u \in F_0$ to v contained in a connected component of $\mathcal{F}$. Let $\mathcal{Q}$ be such that $\mathcal{Q} \cup \mathcal{P}$ is a maximal set of node-disjoint augmenting paths of length $\leqslant k$. By Definition 3.5 we have $|\mathcal{Q}| \leqslant \xi n$. Define the set of *forgotten* vertices as $\bigcup_{Q \in \mathcal{Q}} Q$. Thanks to item 3 in Definition 3.6, the path from u to w has length $\leqslant k$, thus P has length at most k. Recall that P is an augmenting path w.r.t. the graph obtained augmenting M along $\mathcal{P}$ at the end of Step 1. Therefore, P intersects a path in $\mathcal{Q} \cup \mathcal{P}$.

We now argue that that P cannot intersect $P' \in \mathcal{P}$. Suppose by contradiction that it does. Let $P = (P_0 \dots P_\ell)$ and $P' = (P'_0 \dots P'_{\ell'})$. Let P_s be the first (w.r.t. the order induced by P) node where P and P' intersect. We first rule out the case that s is even: for $s = 0$, $P_0 = u \in F_0$ implies that u did not belong to an augmenting path P' in Step 1. Moreover, for $s = 2i > 0$ if $P_{2i} = P'_j$ then $P_{2i-1} = \mathtt{mate}_M(P_{2i}) \in \{P'_{j-1}, P'_{j+1}\}$, where M is the matching obtained at the end of Step 1. Now suppose that s is odd, and hence $P_s \in V_1 \cap P'$. Then $\varphi(P_s)$ is decreased by 1 at the end of Step 1, hence no edge outside of M incident to P_s is eligible in Step 2.

Thus, P must intersect a path in $\mathcal{Q}$. On the other hand $\bigcup_{Q \in \mathcal{Q}} Q$ contains at most $k\xi n$ vertices, so at most $k\xi n$ connected component of $\mathcal{F}$ contain a forgotten edge. Moreover, by item 4 of Definition 3.6 every connected component of $\mathcal{F}$ has size at most 2^k, thus at most $k\xi n 2^k = \gamma/T$ vertices become spurious. □

We say that $B \subseteq V$ is a suitable set of *broken* vertices if all $(u, v) \in (V_0 \backslash B) \times (V_1 \backslash B)$ are 1-feasible.

Lemma 4.3. *After T iterations, there exists a suitable set of broken vertices of size at most γn.*

Proof. First, we prove that every edge $(u, v) \in V_0 \times V_1$, which is 1-feasible at the beginning of Step 1, is also 1-feasible at the end of Step 1. Suppose that (u, v) becomes 1-unfeasible in Step 1. Let M and M' be the matching at the beginning and at the end of Step 1 respectively. Potentials only decrease in Step 1, so in order for (u, v) to become 1-unfeasible w.r.t. M' we must have $(u, v) \in M'$. Moreover, we decrease the potential of v only if $(u, v) \in P$, for some augmenting path P. Thus, at the beginning of Step 1 we had $\varphi(u) + \varphi(v) = c(u, v) + 1$, which implies $\varphi(u) + \varphi(v) = c(u, v)$ at the end of Step 1, thus (u, v) is 1-feasible w.r.t. M', contradiction.

Now, we grow a set of suitable broken vertices B. We initialize $B = \varnothing$ and show that each iteration Step 2 increases the size of B by at most $\gamma n/T$. If $(u, v) \in V_0 \times V_1$ is 1-feasible at the beginning of Step 2 and becomes 1-unfeasible in Step 2, then we must have $u \in \mathcal{F}$ and $v \notin \mathcal{F}$. Indeed, by item 2 in Definition 3.6 if $(u, v) \in M'$ then either both u and v belong to $\mathcal{F}$ or neither of them does. This ensures that the sum of their potentials is unchanged. Else, if $(u, v) \notin M'$ then in order for it to violate 1-feasibility we must increase $\varphi(u)$ by one and not decrease $\varphi(v)$, and this happens only if $u \in \mathcal{F}$ and $v \notin \mathcal{F}$. Item 5 in Definition 3.6 ensures that there exists a vertex cover $U \subseteq V$ for the set of new 1-unfeasible edges with $|U| \leqslant \delta n = \gamma n/T$. We update $B \leftarrow B \cup U$. Thus, after T iterations we have $|B| \leqslant \gamma n$. □

Lemma 4.4. *After T iterations of template algorithm we have have* $|F_0| = |F_1| \leqslant 4\gamma n$.

Proof. Denote with M the final matching obtained by Algorithm 1. Let B be a suitable set of broken vertices with $|B| \leqslant \gamma n$, as in Lemma 4.3. Partition $B = B_M \cup B_F$, where $B_F := B \cap F$ is the set of unmatched vertices in B and B_M is the set of matched vertices in B. Consider the set B'_M of vertices currently matched to vertices in B_M, $B'_M = \{\mathtt{mate}_M(b) \mid b \in B_M\}$. We have $|(B_M \cup B'_M) \cap V_0| = |(B_M \cup B'_M) \cap V_1| \leqslant \gamma n$. Let S be the set of spurious vertices and recall that $|S| \leqslant \gamma n$ by Lemma 4.2. Let $S' \subseteq F_0 \backslash B_F$ such that $|S' \cup (V_0 \cap B_F)| = |S \cup (V_1 \cap B_F)|$. This implies that $|S' \cup (V_0 \cap B_F)| \leqslant |S| + |B| \leqslant 2\gamma n$. Define $A_0 := V_0 \backslash (B \cup B'_M \cup S')$ and $A_1 := V_1 \backslash (B \cup B'_M \cup S)$ and notice that they have the same size. Define $A = A_0 \cup A_1$. Let M' be a perfect matching over A.

The graph $G_A = (A, M \oplus M')$ contains exactly $\ell := |F_0 \cap A_0| = |F_1 \cap A_1|$ node-disjoint paths $P_1 \dots P_\ell$ where P_i starts in $f_0^{(i)} \in F_0 \cap A_0$ and ends in $f_1^{(i)} \in F_1 \cap A_1$. We define the *value* of a path P as

$$\mathcal{V}(P) = \sum_{(u,v) \in M' \cap P} (c(u, v) + 1) - \sum_{(u,v) \in M \cap P} c(u, v).$$

By 1-feasibility of φ we have

$$\mathcal{V}(P_i) \geqslant \sum_{(u,v) \in M' \cap P} (\varphi(u) + \varphi(v)) - \sum_{(u,v) \in M \cap P} (\varphi(u) + \varphi(v)) = \varphi\left(f_0^{(i)}\right) - \varphi\left(f_1^{(i)}\right) = T$$

where the last equality holds by definition of (non-)spurious vertices and Lemma 4.1. Then, we have $Cn \geqslant n + c(M') \geqslant \sum_i^\ell \mathcal{V}(P_i) \geqslant \ell T$. Thus, $\ell \leqslant Cn/T = \gamma n$ and

$$|F_1| = |F_0| \leqslant |F_0 \cap A| + |(B_M \cup B'_M) \cap V_0| + |S' \cup (V_0 \cap B_F)| \leqslant 4\gamma n.$$

□

Let φ be the potential at the end of the execution of Algorithm 1. Denote with $M^{\mathtt{ALG}}$ the final matching obtained by Algorithm 1 and with $M^{\mathtt{OPT}}$ a min-weight perfect matching. Given a matching M, we denote with $M_{[\alpha]}$ the matching obtained from M by removing the αn edges with highest cost.

Lemma 4.5. *We have* $c(M^{\mathtt{ALG}}_{[2\gamma]}) \leqslant c(M^{\mathtt{OPT}})$.

Proof. Let $M^{\mathtt{ALG}}_{\backslash B}$ be the matching obtained from $M^{\mathtt{ALG}}$ by removing all edges incident to vertices in B. Since $|B| \leqslant \gamma n$ we have $c(M^{\mathtt{ALG}}_{[\gamma]}) \leqslant c(M^{\mathtt{ALG}}_{\backslash B})$. Notice that all edges in $M^{\mathtt{ALG}}_{\backslash B}$ are 1-feasible. For each $(u, v) \in M^{\mathtt{ALG}}_{\backslash B}$ we have $c(u, v) = \varphi(u) + \varphi(v)$ and for each $(u, v) \in M^{\mathtt{OPT}}$ we have $\varphi(u) + \varphi(v) \leqslant c(u, v) + 1$. Thus,

$$c(M^{\mathtt{ALG}}_{[\gamma]}) \leqslant c(M^{\mathtt{ALG}}_{\backslash B}) \leqslant \sum_{u \in V} \varphi(u) = \sum_{(u,v) \in M^{\mathtt{OPT}}} \varphi(u) + \varphi(v) \leqslant \sum_{(u,v) \in M^{\mathtt{OPT}}} c(u, v) + 1 = n + c(M^{\mathtt{OPT}}).$$

Now, it is sufficient to notice that, since all edges have costs in $[1/\gamma, C/\gamma - 1]$, removing any γn edges from $M^{\texttt{ALG}}_{[\gamma]}$ decreases its cost by n. Thus, $c(M^{\texttt{ALG}}_{[2\gamma]}) \leqslant c(M^{\texttt{OPT}})$. $\square$

Now, we are ready to prove Theorem 7.

Proof of Theorem 7. Thanks to Lemma 4.5, we know that $c(M^{\texttt{ALG}}_{[2\gamma]}) \leqslant c(M^{\texttt{OPT}})$. Moreover, by Lemma 4.4 we have $|M^{\texttt{ALG}}| = n - |F_0| \geqslant (1 - 4\gamma)n$, thus defining $M^{1-8\gamma}$ as the min-weight matching of size $(1 - 8\gamma)n$, we have $c(M^{1-8\gamma}) \leqslant c(M^{\texttt{ALG}}_{[4\gamma]})$. We are left to prove that the estimate $\hat{c} = \frac{n}{|S|}\Sigma$ returned by Algorithm 1 satisfies $c(M^{\texttt{ALG}}_{[4\gamma]}) \leqslant \hat{c} \leqslant c(M^{\texttt{ALG}}_{[2\gamma]})$. Let S and Σ be defined as in Algorithm 1 and let w be maximum such that $3\gamma|S|$ edges in S have cost $\geqslant w$. If $\alpha_w \cdot n$ is the number of edges in $M^{\texttt{ALG}}$ that cost $\geqslant w$, then using standard Chernoff Bounds arguments we have that, whp, $|\alpha_w - 3\gamma| \leqslant \gamma^2/C$. From now on we condition on this event. Notice that $\frac{(1-\alpha_w)n}{(1-3\gamma)|S|}\Sigma$ is an unbiased estimator of $c(M^{\texttt{ALG}}_{[\alpha_w]})$. Moreover, since all costs are in $[1/\gamma, C/\gamma]$, then $O_{\gamma,C}(\log n)$ samples are sufficient to have $\frac{(1-\alpha_w)n}{(1-3\gamma)|S|}\Sigma$ concentrated, up to a factor $(1 \pm \frac{\gamma^2}{C})$, around $c(M^{\texttt{ALG}}_{[\alpha_w]})$. Hence, assuming that γ is sufficiently small, we have

$$\frac{n}{|S|}\Sigma \in (1 \pm 3\gamma^2/C) \cdot c(M^{\texttt{ALG}}_{[\alpha_w]}) \subseteq c(M^{\texttt{ALG}}_{[\alpha_w]}) \pm 3\gamma n$$

where the last containment relation holds because all costs are $\leqslant C/\gamma$ and so $c(M^{\texttt{ALG}}_{[\alpha_w]}) \leqslant Cn/\gamma$. Since all costs are $\geqslant 1/\gamma$ we have $c(M^{\texttt{ALG}}_{[\alpha_w+3\gamma^2]}) \leqslant c(M^{\texttt{ALG}}_{[\alpha_w]}) - 3\gamma n$ and $c(M^{\texttt{ALG}}_{[\alpha_w-3\gamma^2]}) \geqslant c(M^{\texttt{ALG}}_{[\alpha_w]}) + 3\gamma n$. Thus, picking γ small enough to have $\alpha_w \pm 3\gamma^2 \subseteq [2\gamma, 4\gamma]$ we have

$$c(M^{\texttt{ALG}}_{[4\gamma]}) \leqslant \frac{n}{|S|}\Sigma \leqslant c(M^{\texttt{ALG}}_{[2\gamma]}).$$

Therefore, we have $c(M^{1-8\gamma}) \leqslant \hat{c} \leqslant c(M^{\texttt{OPT}})$ and rescaling γ gives exactly the desired result. $\square$

Observation 8. As in the proof of Theorem 7, define w as the maximum value such that there are at least $3\gamma|S|$ edges with cost $\geqslant w$ in S and define α_w such that exactly $\alpha_w \cdot n$ edges in M have cost $\geqslant w$. We have, whp, $|\alpha_w - 3\gamma| \leqslant \gamma^2/C$, thus for γ small enough $c(M^{\texttt{ALG}}_{[\alpha_w]}) \leqslant c(M^{\texttt{ALG}}_{[2\gamma]}) \leqslant c(M^{\texttt{OPT}})$ and (up to rescaling γ) $|M^{\texttt{ALG}}_{[\alpha_w]}| \geqslant (1 - \gamma)n$. Moreover, given an edge $e \in M^{\texttt{ALG}}$ we can decide whether $e \in M^{\texttt{ALG}}_{[\alpha_w]}$ simply by checking $c(e) \leqslant w$.

5 Implementing the Template in Sublinear Time

In this section we explain how to implement *Step 1* and *Step 2* from the template algorithm in sublinear time.

5.1 From Potential Oracles to Membership Oracles

Throughout this section, we would like to apply Theorem 5 and Theorem 6 on the eligibility graph $G_{\mathcal{E}} = (V, E_{\mathcal{E}})$ and forward graph $G_F = (V, E_F)$. However, we do not have random access to the adjacency matrix of these graphs. Indeed, to establish if $(u, v) \in V_0 \times V_1$ is eligible we need to check the condition $\varphi(u) + \varphi(v) = c(u, v) + 1$ (or $\varphi(u) + \varphi(v) = c(u, v)$). However, we will see that the potential $\varphi(\cdot)$ requires more than a single query to be evaluated. Formally, we assume that we have a potential oracle $\texttt{eval}_\varphi(\cdot)$ that returns the value of $\varphi(\cdot)$ in time t_φ. Whenever checking whether (u, v) is an edge of G_F $(G_\mathcal{E})$ requires to evaluate a condition of the form $\varphi(u) + \varphi(v) = c(u, v) + 1$ (or $\varphi(u) + \varphi(v) = c(u, v)$) we say that we have *potential oracle access* to the adjacency matrix of G_F $(G_\mathcal{E})$ with potential oracle time t_φ. We can think of t_φ as $\tilde{O}(n^{1+\varepsilon})$ and we will later prove that this is (roughly) the case.

Potential functions with constant-size range. If our potential function $\varphi : V \to \mathcal{R}$ has range size $|\mathcal{R}| \leqslant R$ then we say that it is an R-potential. If the eligibility (forward) graph is induced by R-potentials for $R = O(1)$ we can rephrase Theorem 5 and Theorem 6 to work with potential oracle access, without any asymptotic overhead. The following theorem is an analog of Theorem 5 for forward graphs.

Lemma 5.1. *Let $G_F = (V, E_F)$ be a forward graph w.r.t the R-potential φ, let $A \subseteq V$ be a vertex set. Suppose we have a potential oracle $\texttt{eval}_\varphi$ with oracle time t_φ and an membership oracle $\texttt{mem}_A$ with t_A query time. We are given as input constants $0 < \varepsilon \leqslant 0.2$ and $\delta_{\text{in}} > 0$.*

There exists an algorithm ***LargeMatchingForward***$(\varphi, A, \delta_{\text{in}})$ *that preprocesses G_F in $\tilde{O}_R((t_A + t_\varphi + n) \cdot n^{1-\varepsilon})$ time and either returns $\perp$ or constructs a matching oracle $\texttt{match}_M(\cdot)$ for a matching $M \subset G_F[A]$ of size at least $\delta_{\text{out}} n$ where $\delta_{\text{out}} = \frac{1}{2000 \cdot R^{10}} \delta_{\text{in}}^5 = \Theta_{\delta_{\text{in}}, R}(1)$ that has $\tilde{O}_R((t_A + t_\varphi + n)n^{4\varepsilon})$ worst-case query time. If $\mu(G_F[A]) \geqslant \delta_{\text{in}} n$, then $\perp$ is not returned. The guarantee holds with high probability.*

Proof. Without loss of generality, we assume that φ takes values in $[R]$. Suppose that $G_F[A]$ has a matching of size $\delta_{\text{in}} n$. We partition the edges $E_F[A] = E_F \cap (A \times A)$ into R^2 sets $E_{0,0} \dots E_{R-1,R-1}$ such that $(u, v) \in E_{i,j}$ iff $\varphi(u) = i$ and $\varphi(v) = j$. Then, there exist $i, j \in [R]$ such that $G_{i,j} = (V, E_{i,j})$ has a matching of size $\delta_{\text{in}} n / R^2$. Moreover, once we restrict ourselves to $G_{i,j}$, each edge query $(u, v) \in E_{i,j}$ becomes much easier. Indeed, we just need to establish if $i + j = c(u, v) + 1$. In order to restrict ourselves to $G_{i,j}$ it suffices to set $A' = A \cap (\varphi^{-1}(\{i\}) \times \varphi^{-1}(\{j\}))$. Then the membership oracle $\texttt{mem}_{A'}$ runs in time $O(t_A + t_\varphi)$. Hence, using Theorem 5 we can find a matching of size $\delta_{\text{out}} n$, where $\delta_{\text{out}} = \frac{1}{2000 \cdot R^{10}} \delta_{\text{in}}^5$. Algorithmically, we run the algorithm from Theorem 5 R^2 times (once for each pair (i, j)) and halt as soon as the algorithm does not return $\perp$. □

The following is an analog of Theorem 6 for eligibility graphs.

Lemma 5.2. *Let $\varepsilon_{\textit{in}} > 0$ be a sufficiently small constant. Let $\alpha_{k,\gamma}$ and $\beta_{k,\gamma}$ be constants that depend on k and γ and set $\varepsilon_{\textit{out}} := \alpha_{k,\gamma} \cdot \varepsilon_{\textit{in}}$. We have an R-potential oracle $\texttt{eval}_\varphi$ with running time $t_\varphi = \tilde{O}(n^{1+\varepsilon_{\textit{in}}})$, a matching oracle $\texttt{match}_{M^{\textit{in}}}$ with running time $t_{M^{\textit{in}}} = \tilde{O}(n^{1+\varepsilon_{\textit{in}}})$ and an eligibility graph $G_\mathcal{E} = (V, E_\mathcal{E})$ w.r.t. φ and $M^{\textit{in}}$.*

There exists an algorithm ***AugmentEligible***$(\varphi, M^{\textit{in}}, k, \gamma, \varepsilon_{\textit{in}})$ *that runs in $\tilde{O}_{k,\gamma,R}\left(n^{2-\varepsilon_{\textit{in}}}\right)$ time. Further, either it returns an oracle $\texttt{match}_{M^{\textit{out}}}(\cdot)$ with query time $\tilde{O}_{k,\gamma,R}(n^{1+\varepsilon_{\textit{out}}})$, for some matching $M^{\textit{out}}$ in $G_\mathcal{E}$ of size $|M^{\textit{out}}| \geqslant |M^{\textit{in}}| + \beta_{k,\gamma} \cdot n$ (we say that it "succeeds" in this case), or it returns $\perp$. Finally, if the matching $M^{\textit{in}}$ admits a collection of $\gamma \cdot n$ many node-disjoint augmenting paths with length $\leqslant k$ in $G_\mathcal{E}$, then the algorithm succeeds whp.*

Proof. We derive Lemma 5.2 combining Theorem 6 and Lemma 5.1. First, we notice that Theorem 6 says that the algorithm succeeds (whp) whenever there are $\gamma' n$ node-disjoint augmenting paths (NDAP) with length *exactly* $2k' + 1$, while Lemma 5.2 has the weaker requirement that there are at least γn NDAP of length $\leqslant k$. A simple reduction is obtained invoking Theorem 6 with $\gamma' = \gamma / k$ for all k' such that $2k' + 1 \leqslant k$ (notice that all augmenting paths have odd length). In this way, if there exists a collection of γn NDAP of length $\leqslant k$ then there exists a $k' \leqslant (k-1)/2$ such that we have $\gamma' n$ NDAP of length *exactly* $2k'+1$. All guarantees are preserved since we consider both γ and k constants. Now, we are left to address the fact that we do not have random access to the adjacency matrix of $G_{\mathcal{E}}$, but rather potential oracle access.

We notice that, according to Theorem 6, the implementation of `Augment` from [BKS23a] never makes any query to the adjacency matrix besides those performed inside `LargeMatching`. Moreover, Lemma 5.1 implies that `LargeMatchingForward` is not asymptotically slower than `LargeMatching` as long as $R = O(1)$. $\square$

Finally, we observe that in Algorithm 1 each potential is increased (or decreased) at most $T = O_{C,\gamma}(1)$ times. Hence, φ is a R-potential for $R = 2T+1 = O_{C,\gamma}(1)$. Thus, we can consider R a constant when applying Lemma 5.1 or Lemma 5.2.

5.2 Implementing Step 1

In this subsection we implement Step 1 from the template algorithm in sublinear time. Here we assume that we have at our disposal a potential oracle $\texttt{eval}_{\varphi^{\texttt{in}}}$ running in time $t_{\varphi^{\texttt{in}}} = \tilde{O}(n^{1+\varepsilon_{\texttt{in}}})$ and a matching oracle $\texttt{match}_{M^{\texttt{in}}}$ with running time $t_{M^{\texttt{in}}} = \tilde{O}(n^{1+\varepsilon_{\texttt{in}}})$. We will output a potential oracle $\texttt{eval}_{\varphi^{\texttt{out}}}$ running in time $t_{\varphi^{\texttt{out}}} = \tilde{O}(n^{1+\varepsilon_{\texttt{out}}})$ and a matching oracle $\texttt{match}_{M^{\texttt{out}}}$ with running time $t_{M^{\texttt{out}}} = \tilde{O}(n^{1+\varepsilon_{\texttt{out}}})$. We show that there exists a (k,ξ)-QMSNDAP $\mathcal{A}$ such that: the matching $M^{\texttt{out}}$ is obtained from $M^{\texttt{in}}$ by augmenting it along all paths in $\mathcal{A}$; $\varphi^{\texttt{out}}$ is obtained from $\varphi^{\texttt{in}}$ by subtracting 1 to $\varphi^{\texttt{in}}(v)$ for each $v \in V_1 \cap \bigcup_{P\in\mathcal{A}} P$.

Algorithm 2 Implementation of Step 1.

```
Set $k$ and $\gamma$ as in Algorithm 1.
Initialize $M \leftarrow M^{\texttt{in}}$ and $\varepsilon \leftarrow \varepsilon_{\texttt{in}}$.
Repeat until AugmentEligible returns $\perp$:
    1. Let AugmentEligible$(G_{\mathcal{E}}, M, k, \gamma, \varepsilon)$ return $\texttt{match}_{M'}$ with running time $t_{M'} = \tilde{O}(n^{1+\varepsilon'})$.
    2. Update $M \leftarrow M'$ and $\varepsilon \leftarrow \varepsilon'$.
Set $M^{\texttt{out}} \leftarrow M$ and $\varepsilon_{\texttt{out}} \leftarrow \varepsilon$.
```

```
Implement $\texttt{eval}_{\varphi^{\texttt{out}}}(u)$ as follows:
    • If $u \in V_0$, return $\texttt{eval}_{\varphi^{\texttt{in}}}$.
    • Else, $u \in V_1$, set $v \leftarrow \texttt{match}_{M^{\texttt{out}}}(u)$.
    • If $v$ == $\perp$, return $\texttt{eval}_{\varphi^{\texttt{in}}}(u)$.
    • Else, we have $(u,v) \in M^{\texttt{out}}$:
        – If $c(u,v) + 1$ == $\texttt{eval}_{\varphi^{\texttt{in}}}(u) + \texttt{eval}_{\varphi^{\texttt{in}}}(v)$, return $\texttt{eval}_{\varphi^{\texttt{in}}}(u) - 1$.
        – Else, return $\texttt{eval}_{\varphi^{\texttt{in}}}(u)$.
```

Analysis. First, we observe that the algorithm above correctly implements the template, with high probability (all our statements henceforth hold whp). Initialize $\mathcal{A} \leftarrow \varnothing$. For each run of $\texttt{AugmentEligible}(G, M, k, \gamma, \varepsilon)$ we decompose $M \oplus M'$ into a set of augmenting paths $\mathcal{P}$ and a set of alternating cycles $\mathcal{C}$ and we set $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{P}$. When $\texttt{AugmentEligible}(G, M, k, \gamma, \varepsilon)$ returns $\perp$ it means (by Lemma 5.2) that there are at most γn node-disjoint augmenting paths of length $\leqslant k$ that do not intersect $\bigcup \mathcal{A}$. Hence, $\mathcal{A}$ is a (k, ξ)-QMSNDAP . Clearly, $\texttt{match}_{M^{\text{out}}}$ implements the matching obtained from $M^{\texttt{in}}$ by augmenting along the paths in $\mathcal{A}$.

To see that the implementation of $\texttt{eval}_{\varphi^{\text{out}}}(u)$ is correct it is sufficient to notice that in the template algorithm we decrement $\varphi(u)$ iff: (i) $u \in V_1$, and (ii) there exists an augmenting path $P \in \mathcal{A}$ intersecting u. Since every node belongs to at most one path in $\mathcal{A}$ then u is matched in M^{out} and $(u, v) \in M^{\text{out}}$ is an $M^{\texttt{in}}$-eligible edge. Thus, (ii) is equivalent to: (iii) $v = \texttt{match}_{M^{\text{out}}}$ satisfies $\varphi^{\texttt{in}}(v) + \varphi^{\texttt{in}}(u) = c(u, v) + 1$. Finally, we bound $\varepsilon_{\texttt{out}}$ as a function of $\varepsilon_{\texttt{in}}$.

Lemma 5.3. *Step 1 can be implemented in $\tilde{O}(n^{2-\varepsilon})$ time for some constant $\varepsilon > 0$. Moreover, the oracle $\texttt{match}_{M^{out}}$ has running time $t_{M^{out}}$ and the oracle $\texttt{eval}_{\varphi^{out}}$ has running time $t_{\varphi^{out}}$ such that $t_{M^{out}}, t_{\varphi^{out}} = \tilde{O}(n^{1+\varepsilon_{out}})$ and $\varepsilon_{out} = O_{\gamma,k}(\varepsilon_{in})$.*

Proof. Let $\varepsilon > 0$ and $\beta_{k,\gamma}$ as in Lemma 5.2. Algorithm 2 runs $\texttt{AugmentEligible}$ at most $1/\beta_{k,\gamma}+1 = O_{k,\gamma}(1)$ times because the set $\mathcal{A}$ increases by $\beta_{k,\gamma} \cdot n$ after each successful run of $\texttt{AugmentEligible}$. Thus, there can be at most $1/\beta_{k,\gamma}$ successful runs. It is apparent that, by Lemma 5.2, Step 1 can be implemented in $\tilde{O}_{k,\gamma}(n^{2-\varepsilon})$ time.

Now we prove the bound on oracles time. First, we observe that $t_{\varphi^{\text{out}}} = O(t_{M^{\text{out}}} + t_{\varphi^{\text{in}}}) = \tilde{O}(n^{1+\varepsilon_{\text{out}}})$. Moreover, at every iteration we have $\varepsilon' \leqslant \alpha_{k,\gamma} \cdot \varepsilon$, hence $\varepsilon_{\texttt{out}} \leqslant \alpha_{k,\gamma}^{1/\beta_{k,\gamma}+1} \varepsilon_{\texttt{in}} = O_{k,\gamma}(\varepsilon_{\texttt{in}})$. □

5.3 Implementing Step 2

In this subsection we implement Step 2 from Algorithm 1 in sublinear time. Once again, we assume that we have at our disposal a potential oracle $\texttt{eval}_{\varphi^{\text{in}}}$ running in time $t_{\varphi^{\text{in}}} = \tilde{O}(n^{1+\varepsilon_{\text{in}}})$ and a matching oracle $\texttt{match}_{M^{\text{in}}}$ with running time $t_{M^{\text{in}}} = \tilde{O}(n^{1+\varepsilon_{\text{in}}})$. We will output a potential oracle $\texttt{eval}_{\varphi^{\text{out}}}$ running in time $t_{\varphi^{\text{out}}} = \tilde{O}(n^{1+\varepsilon_{\text{out}}})$. We show that there exists a (k, δ)-QMF$\mathcal{F}$ with respect to $M^{\texttt{in}}$ such that $\varphi^{\texttt{out}}(u) = \varphi^{\texttt{in}}(u) + 1$ for each $u \in \mathcal{F} \cap V_0$ and $\varphi^{\texttt{out}}(v) = \varphi^{\texttt{in}}(v) - 1$ for each $v \in \mathcal{F} \cap V_1$.

The execution of Algorithm 3 is represented in Figure 1, where vertices colored in the same way are added to $\mathcal{F}$ during the same iteration.

Analysis. First, we prove that Algorithm 3 implements the template (all guarantees hold whp). Namely, that $\mathcal{F}$ is a (k, δ)-QMF, where k and δ are defined as in Algorithm 1. With a slight abuse of notation, in Algorithm 3 we used $\mathcal{F}$ to denote the set of nodes in the forest. Here, we understand that for each $u \in \mathcal{F}'_t \backslash \mathcal{F}_t$ we have an edge $(u, \texttt{match}_{M_{t+1}}(u))$ and for each $v \in \mathcal{F}''_t \backslash \mathcal{F}'_t$ we have an edge $(v, \texttt{match}_{M^{\text{in}}}(v))$. Let τ be the total number of times $\texttt{LargeMatchingForward}$ runs successfully in Algorithm 3. We will see that $\tau \leqslant k/2$. Notice that $\mathcal{F}_\tau$ is the last forest produced by Algorithm 3 and for each $u \in \mathcal{F}_\tau \backslash F_0$ we add an edge incident to u, thus $\mathcal{F}_\tau$ is a forest with $|F_0|$ connected components, one for each $u \in F_0$. Now we show that $\mathcal{F}_\tau$ is a (k, δ)-QMFw.r.t. $M^{\texttt{in}}$. We refer to the notation of Definition 3.6. Item 1 is clearly satisfied. Item 2 is satisfied because of line 4 in Algorithm 3.

Algorithm 3 Implementation of Step 2.

```
Set $\delta$ as in Algorithm 1.
Initialize $t \leftarrow 0$, $\mathcal{F}_0 \leftarrow F_0$, where $F_0$ is the set of $M^{\texttt{in}}$-unmatched vertices in $V_0$.
Implement $\texttt{mem}_{\mathcal{F}_0}(u)$ as: $\texttt{match}_{M^{\texttt{in}}}(u)$ == $\perp$.
Repeat until LargeMatchingForward returns $\perp$:
    1. $A_t \leftarrow (\mathcal{F}_t \cap V_0) \cup (V_1 \backslash \mathcal{F}_t)$.
    2. Let $\texttt{LargeMatchingForward}(\varphi, A_t, \delta)$ return $\texttt{match}_{M_t}$.
    3. $\mathcal{F}_t' \leftarrow \mathcal{F}_t \cup \{\texttt{match}_{M_t(u)} \mid u \in \mathcal{F}_t\}$
       Implement $\texttt{mem}_{\mathcal{F}_t'}(u)$ as: $\texttt{mem}_{\mathcal{F}_t}(u)$ or $\texttt{match}_{M_t}(u) \in \mathcal{F}_t$.
    4. $\mathcal{F}_t'' \leftarrow \mathcal{F}_t' \cup \{\texttt{match}_{M^{\texttt{in}}}(u) \mid u \in \mathcal{F}_t'\}$
       Implement $\texttt{mem}_{F_t''}(u)$ as: $\texttt{mem}_{\mathcal{F}_t'}(u)$ or $\texttt{match}_{M^{\texttt{in}}}(u) \in \mathcal{F}_t'$.
    5. $\mathcal{F}_{t+1} \leftarrow \mathcal{F}_t''$; $t \leftarrow t+1$.
Implement $\texttt{eval}_{\varphi^{\texttt{out}}}(u)$ as $\texttt{eval}_{\varphi^{\texttt{in}}}(u) + \texttt{mem}_{\mathcal{F}_t}(u) \cdot (-1)^{\mathbb{1}_{u \in V_1}}$
```

Now we show that Item 3 is satisfied. Define $k = 6000(2T+1)^{10}/\delta^5$ as in Algorithm 1 and recall that φ is a $(2T+1)$-potential. Thanks to Lemma 5.1, at each step we increment $|\mathcal{F}|$ by at least $\frac{1}{2000(2T+1)^{10}} \cdot \delta^5 n$. Thus, no more than $\lceil 2000(2T+1)^{10}/\delta^5 \rceil \leqslant k/2$ iterations are performed and we cannot have more than k hops between $u \in \mathcal{F}$ and $v \in F_0$ if u belongs to the connected component of v.

Now we prove that Item 4 is satisfied. At each iteration, the size of each connected component of $\mathcal{F}_t$ at most triples. Indeed, let C be a connected component of $\mathcal{F}$. In step 3 we add to C at most $|C|$ vertices (because we add a vertex for each edge in a matching incident to C) and in step 4 we add to C at most one more vertex for each new vertex added in step 3.

Now we prove that Item 5 is satisfied. Algorithm 3 halts when $\texttt{LargeMatchingForward}(M^{\texttt{in}}, (\mathcal{F} \cap V_0) \cup (V_1 \backslash \mathcal{F}), \delta)$ returns $\perp$. This may only happen when there is no matching between of $\mathcal{F} \cap V_0$ and $V_1 \backslash \mathcal{F}$ of size δn. This implies that there exists a vertex cover of size $\leqslant \delta n$. Moreover, this is a vertex cover for the whole $E_{\mathcal{E}} \cap (\mathcal{F} \times V \backslash \mathcal{F})$ because all edges in $E_{\mathcal{E}} \cap V_1 \times V_0$ are in $M^{\texttt{in}}$ and by Item 2 have both endpoints either in $\mathcal{F}$ or in $V \backslash \mathcal{F}$.

It is easy to check that $\varphi^{\texttt{out}}(u) = \varphi^{\texttt{in}}(u) + 1$ for each $u \in \mathcal{F} \cap V_0$ and $\varphi^{\texttt{out}}(v) = \varphi^{\texttt{in}}(v) - 1$ for each $v \in \mathcal{F} \cap V_1$.

Lemma 5.4. *Step 2 can be implemented in $\tilde{O}(n^{2-\varepsilon})$ time for some constant $\varepsilon > 0$. Moreover, the oracle* $\texttt{eval}_{\varphi_{out}}$ *has running time $t_{\varphi^{out}} = \tilde{O}(n^{1+\varepsilon_{out}})$ and $\varepsilon_{out} = O_{k,\delta}(\varepsilon_{in})$.*

Proof. For $s = 0 \dots \tau$ denote with $\varepsilon_s > 0$ a constant such that $t_{A_s} = \tilde{O}(n^{1+\varepsilon_s})$, where t_{A_s} is the running time of $\texttt{mem}_{A_s}$. Notice that $\varepsilon_0 = \varepsilon_{\texttt{in}}$. At step s we choose $\hat{\varepsilon}_s := 2\varepsilon_s$ as the ε parameter in Lemma 5.1. This implies that $\texttt{LargeMatchingForward}$ runs in $\tilde{O}(n^{1+\varepsilon_s} \cdot n^{1-\hat{\varepsilon}_s}) = \tilde{O}(n^{2-\varepsilon_s})$ time. We have already proved that $\tau \leqslant k$, thus Algorithm 2 takes $\tilde{O}(n^{2-\varepsilon})$ time in total, where $\varepsilon := \min_{s \in [0,\tau]} \varepsilon_s = \varepsilon_{\texttt{in}}$.

Denote with $t_{\mathcal{F}}$ the query time of $\texttt{mem}_{\mathcal{F}}$. For each s, we have $t_{\mathcal{F}_{s+1}} = t_{M_{s+1}} + t_{M^{\texttt{in}}} + t_{\mathcal{F}_s}$. Thanks to Lemma 5.1 we have $t_{M_{s+1}} = \tilde{O}((t_{A_s} + t_{\varphi^{\texttt{in}}} + n)n^{4\hat{\varepsilon}_s})$. Moreover, $t_{A_s} = t_{\mathcal{F}_s}$. Thus,

$t_{\mathcal{F}_{s+1}} = (t_{\mathcal{F}_s} + t_{\varphi^{\texttt{in}}} + n)n^{8\varepsilon_s} + t_{M^{\texttt{in}}} + t_{\mathcal{F}_s}$. Since, $t_{\mathcal{F}_0} = t_{\varphi^{\texttt{in}}} = t_{M^{\texttt{in}}} = \tilde{O}(n^{1+\varepsilon_{\texttt{in}}})$ we have $t_{\varphi^{\texttt{out}}} = t_{\varphi^{\texttt{in}}} + t_{\mathcal{F}_\tau} = \tilde{O}(n^{1+9^\tau \cdot \varepsilon_{\texttt{in}}}) = \tilde{O}_k(n^{1+O_{k,\delta}(\varepsilon_{\texttt{in}})})$. □

5.4 Implementing the Template Algorithm

We can put together the results proved in the previous subsections and show that Algorithm 1 can be implemented in sublinear time.

Theorem 9. *There exists a constant $\varepsilon > 0$ such that Algorithm 1 can be implemented in time $O(n^{2-\varepsilon})$. Moreover, using the notation in Observation 8, we can return a matching oracle $\texttt{match}_{M^{\textit{ALG}}_{[\alpha_w]}}$ running in time $O(n^{1-\varepsilon})$ such that $M^{\textit{ALG}}_{[\alpha_w]}$ satisfies $|M^{\textit{ALG}}_{[\alpha_w]}| \geqslant (1-\gamma)n$ and $c(M^{\textit{ALG}}_{[\alpha_w]}) \leqslant c(M^{\textit{OPT}})$ and $\texttt{match}_{M^{\textit{ALG}}_{[\alpha_w]}}$.*

Proof. Algorithm 1 runs $T = O_{C,\gamma}(1)$ iterations, and a single iterations consists of Step 1 and Step 2. At iteration s denote with $\varepsilon^{(s)}_{\texttt{in}}$ the value of $\varepsilon_{\texttt{in}}$ for Step 1 input (or, equivalently, the value of $\varepsilon_{\texttt{out}}$ for Step 2 output at iteration $s-1$) and with $\varepsilon^{(s)}_{\texttt{out}}$ the value of $\varepsilon_{\texttt{out}}$ for Step 1 output (or, equivalently, the value of $\varepsilon_{\texttt{in}}$ for Step 2 input at iteration s). Every time we run either Step 1 or Step 2, the value of $\varepsilon_{\texttt{out}}$ is at most some constant factor larger than $\varepsilon_{\texttt{in}}$. This translates into $\varepsilon^{(s)}_{\texttt{in}} = O_{C,k,\gamma}(\varepsilon^{(s-1)}_{\texttt{out}})$ and $\varepsilon^{(s)}_{\texttt{out}} = O_{C,k,\gamma}(\varepsilon^{(s)}_{\texttt{in}})$. Thus, after T iterations $\varepsilon^{(T)}_{\texttt{out}}$ is arbitrarily small, provided that $\varepsilon^{(0)}_{\texttt{in}}$ is small enough. To conclude, we notice that the initial matching is empty and the initial potential is identically 0, so the first membership oracle and potential oracle run in linear linear, thus we can set $\varepsilon^{(0)}_{\texttt{in}}$ arbitrarily small. Finally, let $M^{\texttt{ALG}}$ be the last matching computed by Algorithm 1. We have at our disposal a matching oracle $\texttt{match}_{M^{\texttt{ALG}}}$ running in time $\tilde{O}(n^{1+\varepsilon^{(T+1)}_{\texttt{in}}})$, so we can easily sample the a S of $O(\log n)$ edges from $M^{\texttt{ALG}}$ in time $\tilde{O}(n^{1+\varepsilon^{(T)}_{\texttt{out}}})$. This conclude the implementation of Algorithm 1.

Moreover, we compute w as the largest value such that at least $3\gamma|S|$ edges in S have cost $\geqslant w$ and define α_w such that exactly $\alpha_w \cdot n$ edges in $M^{\texttt{ALG}}$ have cost $\geqslant w$. Then, we implement a matching oracle $\texttt{match}_{M^{\texttt{ALG}}_{[\alpha_w]}}$ for $M^{\texttt{ALG}}_{[\alpha_w]}$ running in time $\tilde{O}(n^{1+\varepsilon^{(T)}_{\texttt{out}}})$ as follows: given $u \in V_0 \cup V_1$ we set $v \leftarrow \texttt{match}_{M^{\texttt{ALG}}}(u)$; if $c(u,v) < w$ then we return v, else we return $\perp$. Thanks to Observation 8, we have $|M^{\texttt{ALG}}_{[\alpha_w]}| \geqslant (1-\gamma)n$ and $c(M^{\texttt{ALG}}_{[\alpha_w]}) \leqslant c(M^{\texttt{OPT}})$. □

6 Proof of our Main Theorems

In this section we piece things together and prove Theorem 3. Then, we use Theorem 3 to prove Theorem 1, Corollary 1.1 and Theorem 2.

6.1 Proof of Theorem 3

In this subsection we strengthen Theorem 7, extend its scope to arbitrary costs and combine it with Theorem 9 to obtain Theorem 3. We restate the latter for convenience.

Theorem 3. *For each constants $0 \leqslant \alpha < \beta \leqslant 1$ there exists a constant $\varepsilon > 0$ and an algorithm running in time $O(n^{2-\varepsilon})$ with the following guarantees.*

The algorithm has adjacency-matrix access to an undirected, bipartite graph $G = (V_0 \cup V_1, E)$ and random access to the edge-cost function $c : E \to \mathbb{R}^+$. The algorithm returns $\hat{c}$ such that, whp,

$$c(M^\alpha) \leqslant \hat{c} \leqslant c(M^\beta)$$

where M^α is a minimum-weight matching of size αn and M^β is a minimum-weight matching of size βn.

Moreover, the algorithm returns a matching oracle data structure that, given a vertex u returns, in $n^{1+f(\varepsilon)}$ time, an edge $(u, v) \in \hat{M}$ or $\perp$ if $u \notin V(\hat{M})$, where $f(\varepsilon) \to 0$ when $\varepsilon \to 0$. The matching $\hat{M}$ satisfies $\alpha n \leqslant |\hat{M}| \leqslant \beta n$ and $c(M^\alpha) \leqslant c(\hat{M}) \leqslant c(M^\beta)$.

Roadmap of the proof. Theorem 7 works only for weights in $[1, C]$. In order to reduce to that case, we need to find a *characteristic cost* $\bar{w}$ of min-weight matchings with size in $[\alpha n, \beta n]$. Then, we round every cost to a multiple of $\frac{1}{2}\gamma^2 \bar{w}$, where γ is a small constant. We show that, thanks to certain properties of the characteristic cost $\bar{w}$, the approximation error induced by rounding the costs is negligible. Finally, we pad each size of the bipartition with dummy vertices to reduce the problem of finding a matching of approximate size βn to that of finding an *approximate perfect matching*, which is addressed in Theorem 7.

Notation. Similarly to Theorem 3, we denote with M^ξ the min-weight matching of size ξn in G. Likewise, we will define a graph $\bar{G}$ and denote with $\overline{M}^\xi$ the min-weight matching of size ξn in $\bar{G}$. As in Section 5, given a matching M, we denote with $M_{[\delta]}$ the matching obtained from M by removing the δn most expensive edges. We denote with $\mu(M)$ the cost of the most expensive edge in M. Given $w \geqslant 0$, we denote with $G_{\leqslant w}$ the graph of edges which cost $\leqslant w$. Throughout this subsection, fix a constant $0 < \gamma < (\beta - \alpha)/4$.

Reduction from arbitrary weights to $[1, C]$. The next technical lemma shows that, if we can can solve an easier version of the problem in Theorem 3 where we allow an additive error $\gamma^2 \bar{w} n$ on an instance where $\bar{w}$ is an upper bound for the cost function; then we can also solve the problem in Theorem 3. This reduction is achieved by finding a suitable characteristic cost $\bar{w}$ in sublinear time and running the aforementioned algorithm on $G_{\leqslant \bar{w}}$.

Lemma 6.1. *Suppose that there exists an algorithm that takes as input a bipartite graph $\bar{G} = (\bar{V}_0 \cup \bar{V}_1, \bar{E})$ endowed with a cost function $\bar{c} : \bar{E} \to [0, \bar{w}]$, outputs an estimate $\hat{c}$ and a matching oracle* $\texttt{match}_{\hat{M}}$ *such that (whp) $\hat{c}$ satisfies*

$$\bar{c}(\overline{M}^\alpha) \leqslant \hat{c} \leqslant \bar{c}(\overline{M}^{\alpha+\gamma}) + \gamma^2 \bar{w} n$$

while $\hat{M}$ satisfies $|\hat{M}| \geqslant \alpha n$ and $c(\hat{M}) \leqslant \bar{c}(\overline{M}^{\alpha+\gamma}) + \gamma^2 \bar{w} n$. Suppose also that such algorithm runs in time $O(\bar{n}^{2-\varepsilon})$ and $\texttt{match}_{\hat{M}}$ *runs in time $O(\bar{n}^{1+\varepsilon})$ for some $\varepsilon > 0$, where $\bar{n} = |\bar{V}_0| = |\bar{V}_1|$.*

Then, there exists an algorithm that takes as input a bipartite graph $G = (V_0 \cup V_1, E)$ endowed with a cost function $c : E \to \mathbb{R}^+$, outputs an estimate $\hat{c}$ and a matching oracle $\texttt{match}_{\hat{M}}$ *such that (whp) $\hat{c}$ satisfies*

$$C(M^\alpha) \leqslant \hat{c} \leqslant c(M^\beta)$$

while $\hat{M}$ satisfies $|\hat{M}| \geqslant \alpha n$ and $c(\hat{M}) \leqslant c(M^\beta)$. Moreover, such algorithm runs in time $O(n^{2-\varepsilon})$ and $\texttt{match}_{\hat{M}}$ *runs in time $O(n^{1+\varepsilon})$ for some $\varepsilon > 0$, where $n = |V_0| = |V_1|$.*

Proof. First, we show how to compute, in time $O(n^{2-\varepsilon})$, a value $\bar{w}$ such that:

(i) $\gamma \cdot \bar{w} \leqslant \mu(M^{\beta}_{[\gamma]})$

(ii) $\bar{w} \geqslant \mu(M^{\alpha+3\gamma}_{[2\gamma]})$.

We sample $s = \Theta(n \log n/\gamma)$ edges from G uniformly at random. Let $w_1 \leqslant w_2 \leqslant \cdots \leqslant w_s$ be their costs. Recall that Theorem 4 allows us to compute the size of a maximal-cardinality matching (MCM) of the graph G, up to a γn-additive approximation, in time $O(n^{2-\Omega_\gamma(1)})$. Denote that algorithm with `ApproximateMatching`(G, γ). Using binary search, we find the largest cost w_i such that `ApproximateMatching`$(G_{\leqslant \gamma w_i}, \gamma)$ returns an estimated MCM size $< (\beta - 2\gamma)n$. Then, we set $\bar{w} := w_i$.

We prove that property (i) holds. Suppose that $\mu(M^{\beta}_{[\gamma]}) < \gamma w_i$. Then, in $G_{\leqslant \gamma w_i}$ there exists a matching of size $(\beta - \gamma)n$, therefore `ApproximateMatching`$(G_{\leqslant \gamma w_i}, \gamma)$ finds a matching of size $\geqslant (\beta - 2\gamma)n$ whp, contradiction.

We prove that property (ii) holds. First, we prove that $w_{i+1} \geqslant \mu(M^{\alpha+3\gamma}_{[\gamma]})$. Indeed, suppose the reverse (strict) inequality holds. Then, for each matching M of size $(\beta - 2\gamma)n$ we have

$$c(M) \geqslant c(M^{\beta-2\gamma}) \geqslant c(M^{\alpha+3\gamma}) > \gamma \cdot w_{i+1} n$$

which implies that there exists $e \in M$ such that $c(e) > \gamma \cdot w_{i+1}$. However, this cannot hold for each matching M of size $(\beta - 2\gamma)n$ because `ApproximateMatching`$(G_{\leqslant \gamma w_{i+1}}, \gamma)$ returned (whp) a matching (oracle) of size $\geqslant (\beta - 2\gamma)n$. Contradiction. Therefore, we have $w_{i+1} \geqslant \mu(M^{\alpha+3\gamma}_{[\gamma]})$. However, since we sampled $\Theta(n \log n/\gamma)$ edges, then for each i there are, whp, at most $\gamma \cdot n$ edges which cost w satisfies $w_i < w < w_{i+1}$. Hence, removing γn more edges from $M^{\alpha+3\gamma}_{[\gamma]}$ we obtain $w_i \geqslant \mu(M^{\alpha+3\gamma}_{[2\gamma]})$.

We define $\bar{G} := G_{\leqslant \bar{w}}$ and $\bar{c} = c|_{\bar{E}}$, run the algorithm in the premise of the lemma on $\bar{G}$ and $\bar{c}$ and let $\hat{c}$, `match`$_{\hat{M}}$ be its outputs. It is apparent that this reduction takes $O(n^{2-\varepsilon})$ for some $\varepsilon > 0$.

Since $\bar{G}$ is a subgraph of G, we have $c(M^{\alpha}) \leqslant c(\overline{M}^{\alpha})$. Moreover, conditions (i) and (ii), together with $5\gamma \leqslant \beta - \alpha$ imply

$$c(\overline{M}^{\alpha+\gamma}) \leqslant c(M^{\alpha+3\gamma}_{[2\gamma]}) \leqslant c(M^{\alpha+3\gamma}) \leqslant c(M^{\beta-\gamma}) \leqslant c(M^{\beta}_{[\gamma]}) \leqslant c(M^{\beta}) - \gamma n \cdot \mu(M^{\beta}_{[\gamma]}) \leqslant c(M^{\beta}) - \gamma^2 \bar{w} n.$$

Thus, $c(\overline{M}^{\alpha}) \leqslant \hat{c} \leqslant c(\overline{M}^{\alpha+\gamma}) + \gamma^2 \bar{w} n$ implies $c(M^{\alpha}) \leqslant \hat{c} \leqslant c(M^{\beta})$ and $c(\hat{M}) \leqslant c(\overline{M}^{\alpha+\gamma}) + \gamma^2 \bar{w} n$ implies $\hat{c} \leqslant c(M^{\beta})$. ☐

The following lemma shows how to reduce from real-values costs in $[0, w]$ (where possibly $w = \omega(1)$) to the more tame case where costs are integers in $[1, C]$. This reduction is achieved via rounding.

Lemma 6.2. *Suppose that there exists an algorithm that takes as input a bipartite graph $\bar{G} = (\bar{V}_0 \cup \bar{V}_1, \bar{E})$ endowed a cost function $\bar{c} : E \to [1, C]$, with $C = O(1)$, returns an estimate $\hat{c}$ and a matching oracle* `match`$_{\hat{M}}$ *such that (whp) $\hat{c}$ satisfies*

$$\bar{c}(\overline{M}^{\alpha}) \leqslant \hat{c} \leqslant \bar{c}(\overline{M}^{\beta})$$

while $\hat{M}$ satisfies $|\hat{M}| \geqslant \alpha n$ and $\bar{c}(\hat{M}) \leqslant \bar{c}(\overline{M}^{\beta})$. Suppose also that such algorithm runs in time $O(\bar{n}^{2-\varepsilon})$ and `match`$_{\hat{M}}$ *runs in time $O(\bar{n}^{1+\varepsilon})$ for some $\varepsilon > 0$, where $\bar{n} = |\bar{V}_0| = |\bar{V}_1|$.*

Then, there exists an algorithm that takes as input a bipartite graph $G = (V_0 \cup V_1, E)$ endowed a cost function $c : E \to [0, w]$ (possibly $w = \omega(1)$), returns an estimate $\tilde{c}$ and a matching oracle $\texttt{match}_{\tilde{M}}$ such that (whp) $\tilde{c}$ satisfies

$$c(M^\alpha) \leqslant \tilde{c} \leqslant c(M^\beta) + \gamma^2 wn$$

while $\tilde{M}$ satisfies $|\tilde{M}| \geqslant \alpha n$ and $c(\tilde{M}) \leqslant c(M^\beta) + \gamma^2 wn$. Moreover, such algorithm runs in time $O(n^{2-\varepsilon})$ and $\texttt{match}_{\hat{M}}$ runs in time $O(n^{1+\varepsilon})$ for some $\varepsilon > 0$, where $n = |V_0| = |V_1|$.

Proof. We define $\bar{c}(e) = \left\lceil \frac{2c(e)}{\gamma^2 w} \right\rceil + 1$. Then, the maximum value of $\bar{c}$ on $\bar{G}$ is $C := 2/\gamma^2 + 2 = O(1)$. We set $\bar{G} = G$ and run the algorithm in the premise of the lemma on $\bar{G}$ and $\bar{c}$. Let $\hat{c}$, $\texttt{match}_{\hat{M}}$ be its outputs. We define $\tilde{c} := \frac{1}{2}\gamma^2 w \cdot \hat{c}$ and $\tilde{M} := \hat{M}$. The definition of $\bar{c}$ implies that, for each edge e in $\bar{G}$, $c(e) \leqslant \frac{1}{2}\gamma^2 w \cdot \bar{c}(e) \leqslant c(e) + \gamma^2 w$. Hence,

$$\frac{1}{2}\gamma^2 w \cdot \bar{c}(\overline{M}^\alpha) \geqslant c(\overline{M}^\alpha) \geqslant c(M^\alpha)$$

and

$$\frac{1}{2}\gamma^2 w \cdot \bar{c}(\overline{M}^\beta) \leqslant \frac{1}{2}\gamma^2 w \cdot \bar{c}(M^\beta) \leqslant c(M^\beta) + \gamma^2 wn.$$

Therefore, $\bar{c}(\overline{M}^\alpha) \leqslant \hat{c} \leqslant \bar{c}(\overline{M}^\beta)$ implies $c(M^\alpha) \leqslant \tilde{c} \leqslant c(M^\beta) + \gamma^2 wn$ and $\bar{c}(\hat{M}) \leqslant \bar{c}(\overline{M}^\beta)$ implies $c(\hat{M}) \leqslant \frac{1}{2}\gamma^2 w \cdot \bar{c}(\hat{M}) \leqslant \frac{1}{2}\gamma^2 w \cdot \bar{c}(\overline{M}^\beta) \leqslant c(M^\beta) + \gamma^2 wn.$ □

Reduction from size-βn matching to perfect matching. The following lemma shows that, if we can approximate the min-weight of a perfect matching (allowing δn outliers), then we can approximate the min-weight of a size-βn matching (allowing $(\beta - \alpha)n$ outliers). This reduction is achieved by padding the original graph with dummy vertices.

Lemma 6.3. *Suppose that, for each $\delta > 0$, there exists an algorithm that takes as input a bipartite graph $\bar{G} = (\bar{V}_0 \cup \bar{V}_1, \bar{E})$ endowed a cost function $\bar{c} : E \to [1, C]$, with $C = O(1)$, returns an estimate $\hat{c}$ and a matching oracle $\texttt{match}_{\hat{M}}$ such that (whp) $\hat{c}$ satisfies*

$$\bar{c}(\overline{M}^{1-\delta}) \leqslant \hat{c} \leqslant \bar{c}(\overline{M}^1)$$

while $\hat{M}$ satisfies $|\hat{M}| \geqslant (1-\delta)n$ and $c(\hat{M}) \leqslant \bar{c}(\overline{M}^1)$. Suppose also that such algorithm runs in time $O(\bar{n}^{2-\varepsilon})$ and $\texttt{match}_{\hat{M}}$ runs in time $O(\bar{n}^{1+\varepsilon})$ for some $\varepsilon > 0$, where $\bar{n} = |\bar{V}_0| = |\bar{V}_1|$.

Then, for each $0 \leqslant \alpha < \beta \leqslant 1$ there exists an algorithm that takes as input a bipartite graph $G = (V_0 \cup V_1, E)$ and $c : E \to [1, C]$, returns an estimate $\tilde{c}$ and a matching oracle $\texttt{match}_{\tilde{M}}$ such that (whp) $\tilde{c}$ satisfies

$$c(M^\alpha) \leqslant \tilde{c} \leqslant c(M^\beta)$$

while $\tilde{M}$ satisfies $|\tilde{M}| \geqslant \alpha n$ and $c(\tilde{M}) \leqslant c(M^\beta)$. Moreover, such algorithm runs in time $O(n^{2-\varepsilon})$ and $\texttt{match}_{\tilde{M}}$ runs in time $O(n^{1+\varepsilon})$ for some $\varepsilon > 0$, where $n = |V_0| = |V_1|$.

Proof. Fix a constant $0 < \xi \leqslant (\alpha - \beta)/2$. We construct $\bar{G}$, starting from $\bar{G} = G$ and $\bar{c} = c$, as follows. We add a set of $(1 - \beta + \xi)n$ dummy vertices on each side of the bipartition: $\bar{V}_0 = V_0 \cup D_0$ and $\bar{V}_1 = V_1 \cup D_1$. Add an edge (d_0, v_1) to E for each $(d_0, v_1) \in D_0 \times V_1$ and set $\bar{c}(d_0, v_1) = 1$. Do the same for each $(v_0, d_1) \in V_0 \times D_1$. Notice that we construct both the adjacency matrix and the cost function implicitly, because an explicit construction would take $\Omega(n^2)$ time. We have $\bar{n} := |\bar{V}_0| = |\bar{V}_1| = (2 - \beta + \xi)n$.

Set $\delta = \frac{\xi n}{2\bar{n}}$. Run the algorithm in hypothesis on $\bar{G}$ and let $\hat{c}$ and $\mathtt{match}_{\hat{M}}$ be its outputs. We set $\tilde{c} = \hat{c} - (2 - 2\beta + \xi)n$. We set $\tilde{M} := \hat{M} \cap (V_0 \times V_1)$ and implement $\mathtt{match}_{\tilde{M}}(u)$ as follows. Let $v \leftarrow \mathtt{match}_{\hat{M}}(u)$. If $v = \bot$, return $\bot$. If either u or v is dummy, return $\bot$; else return v. It is easy to see that $\overline{M}^1 \cap (V_0 \times V_1)$ is a min-weight matching of size $(\beta - \xi)n$ in G, hence

$$\bar{c}(\overline{M}^1) = c(M^{\beta-\xi}) + |D_0| + |D_1| = c(M^{\beta-\xi}) + 2(1 - \beta + \xi)n \leqslant c(M^{\beta}) + (2 - 2\beta + \xi)n.$$

On the other hand, in $\overline{M}^{1-\delta}$ at most $2\delta\bar{n} = \xi n$ dummy vertices are left unmathced, so at least $(2 - 2\beta + \xi)n$ dummy vertices are matched in $\overline{M}^{1-\delta}$. Moreover, $\overline{M}^{1-\delta} \cap (V_0 \times V_1)$ is a matching in G of size $\geqslant n - (1 - \beta + \xi)n - \delta\bar{n} = (\beta - 2\xi)n$. Hence,

$$\bar{c}(\overline{M}^{1-\delta}) \geqslant c(M^{\beta-2\xi}) + (2 - 2\beta + \xi)n \geqslant c(M^{\alpha}) + (2 - 2\beta + \xi)n.$$

Thus, $\bar{c}(\overline{M}^{1-\delta}) \leqslant \hat{c} \leqslant \bar{c}(\overline{M}^1)$ implies $c(M^{\alpha}) \leqslant \tilde{c} \leqslant c(M^{\beta})$.

Now we prove the bounds on $\tilde{M}$. We have that since $|\hat{M}| \geqslant (1 - \delta)\bar{n}$, then at most $2\delta\bar{n} = \xi n$ dummy vertices are left unmatched in $\hat{M}$ and so

$$\begin{aligned} c(\tilde{M}) &\leqslant \bar{c}(\hat{M}) - (2(1 - \beta + \xi)n - 2\delta\bar{n}) \\ &\leqslant \bar{c}(\overline{M}^1) - (2 - 2\beta + \xi)n \\ &= c(M^{\beta-\xi}) + 2(1 - \beta - \xi)n - (2 - 2\beta + \xi)n \\ &= c(M^{\beta-\xi}) + \xi n \leqslant c(M^{\beta}). \end{aligned}$$

Moreover, $|\tilde{M}| \geqslant (1 - \delta)\bar{n} - 2(1 - \beta + \xi)n = (\beta - \frac{3}{2}\xi)n \geqslant \alpha n$. □

Finally, we can prove Theorem 3.

Proof of Theorem 3. We notice that combining Theorem 7 and Theorem 9 we have a sublinear implementation of Algorithm 1 that takes a graph bipartite graph $G = (V_0 \cup V_1, E)$ and a cost function $c : E \to [1, C]$ as input, outputs an estimate $\hat{c}$ and a matching oracle $\mathtt{match}_{\hat{M}}$. The estimate $\hat{c}$ satisfies $c(M^{1-\delta}) \leqslant \hat{c} \leqslant c(M^1)$, $\hat{M}$ satisfies $|\hat{M}| \geqslant (1 - \delta)n$ and $c(\hat{M}) \leqslant c(M^{\mathtt{OPT}})$. Moreover, such algorithm runs in time $O(n^{2-\varepsilon})$ and $\mathtt{match}_{\hat{M}}$ runs in time $O(n^{1+\varepsilon})$ for some $\varepsilon > 0$.

Then, combining Lemma 6.3, Lemma 6.2 and Lemma 6.1 we obtain an algorithm that takes as input a bipartite graph $G = (V_0 \cup V_1, E)$ endowed with a cost function $c : E \to \mathbb{R}^+$, outputs an estimate $\hat{c}$ and a matching oracle $\mathtt{match}_{\hat{M}}$ such that (whp) $c(M^{\alpha}) \leqslant \hat{c} \leqslant c(M^{\beta})$, $|\hat{M}| \geqslant \alpha n$, and $c(\hat{M}) \leqslant c(M^{\beta})$. Moreover such algorithm runs in time $O(n^{2-\varepsilon})$ and $\mathtt{match}_{\hat{M}}$ runs in time $O(n^{1+\varepsilon})$ for some $\varepsilon > 0$. □

6.2 Proof of Theorem 1 and Corollary 1.1

Since Corollary 1.1 is more general than Theorem 1 we simply prove the former.

Corollary 1.1. *Suppose we have sample access to two distributions μ, ν over metric space $(\mathcal{M}, d_{\mathcal{M}})$ satisfying $d(\cdot, \cdot) \in [0, 1]$ and query access to d. Suppose further that there exist μ', ν' with support size n such that* $\mathrm{EMD}(\mu, \mu'), \mathrm{EMD}(\nu, \nu') \leqslant \xi$, *for some $\xi > 0$.*

For each constant $\gamma > 0$ there exists a constant $\varepsilon > 0$ and an algorithm running in time $O(n^{2-\varepsilon})$ that outputs $\widehat{\text{EMD}}$ such that

$$\widehat{\text{EMD}} \in [\text{EMD}(\mu, \nu) \pm (4\xi + \gamma)].$$

Moreover, such algorithm takes $\tilde{O}(n)$ samples from μ and ν.

Fix a constant $\gamma > 0$. From each probability distribution μ, ν we sample (with replacement) a multi-set of $m = \Theta(n \log(n))$ points. We use V_μ, V_ν to denote the respective multi-sets, and $\hat{\mu}, \hat{\nu}$ to denote the empirical distributions of sampling a random point from V_μ, V_ν. Let $\mathcal{T}_\mu, \mathcal{T}_\nu$ be the transport plans realizing $\text{EMD}(\mu, \mu')$ and $\text{EMD}(\nu, \nu')$ respectively. Namely, $\mathcal{T}_\mu$ is a coupling between μ and μ' such that $\text{EMD}(\mu, \mu') = E_{(x,y)\sim\mathcal{T}_\mu}[d(x, y)]$ and likewise for $\mathcal{T}_\nu$. For each sample x in $\hat{\mu}$ we sample $x' \sim \mathcal{T}(x, \cdot)$ and let V'_μ be the multi-set of samples x' for $x \in V_\mu$. Define V'_ν similarly. Let $\hat{\mu}'$ and $\hat{\nu}'$ be the empirical distributions of sampling a random point from V'_μ and V'_ν.

Lemma 6.4. $\text{EMD}(\hat{\mu}, \hat{\mu}') \leqslant \xi + \gamma$ *with high probability.*

Proof. $\text{E}[\text{EMD}(\hat{\mu}, \hat{\mu}')] \leqslant \text{E}\left[\frac{1}{m} \cdot \sum_{x\in V_\mu} d(x, x')\right] = \text{E}_{(x,x')\sim\mathcal{T}_\mu}[d(x, x')] = \text{EMD}(\mu, \mu') \leqslant \xi$. Moreover, $\text{Var}_{(x,x')\sim\mathcal{T}_\mu}[d(x, x')] \leqslant 1$, thus $m = \Theta(n \log n)$ ensures $\text{EMD}(\hat{\mu}, \hat{\mu}') \leqslant \xi + \gamma$ whp. □

Lemma 6.5. $\text{EMD}(\hat{\mu}', \mu') \leqslant \text{TV}(\hat{\mu}', \mu') \leqslant \gamma$ *with high probability.*

Proof. First, we observe that V'_μ is distributed as a multi-set of m samples from μ'. For any point x' with at least $(\gamma/4n)$-mass in μ', we expect $\Omega(\log n)$ samples of x' in V'_μ, so by Chernoff bound we have that with high probability the number of samples of x' concentrates to within $(1 \pm \gamma/4)$-factor of its expectation. Furthermore, with high probability at most $(\gamma/2)$-fraction of the samples correspond to points with less than $(\gamma/4n)$-mass in the original distribution. Thus overall, the empirical distribution $\hat{\mu}'$ is within γ TV distance of μ'. Finally, $\text{EMD}(\hat{\mu}', \mu') \leqslant \text{TV}(\hat{\mu}', \mu')$ beacuse $d(\cdot, \cdot) \in [0, 1]$. □

Lemma 6.6. $\text{EMD}(\mu, \hat{\mu}) \leqslant \xi + 2\gamma$ *with high probability.*

Proof. Combining Lemma 6.4, Lemma 6.5 we obtain the following

$$\text{EMD}(\mu, \hat{\mu}) \leqslant \text{EMD}(\mu, \mu') + \text{EMD}(\mu', \hat{\mu}') + \text{EMD}(\hat{\mu}', \hat{\mu}) \leqslant 2\xi + 2\gamma.$$

□

Proof of Corollary 1.1. We consider the bipartite graph with a vertex for each point in V_μ, V_ν and edge costs induced by $d(\cdot, \cdot)$. We apply the algorithm guaranteed by Theorem 3 to find an estimate of the min-weight matching over between $(1 - \gamma)m$ and m vertices. We return the cost estimate $\widehat{\text{EMD}}$ on the bipartite graph (normalized by dividing by m). Theorem 3 guarantees that $\widehat{\text{EMD}} \in [\text{EMD}(\hat{\mu}, \hat{\nu}) \pm \gamma]$. Then using triangle inequality on EMD, as well as Lemma 6.6 on both μ and ν we obtain

$$\left|\widehat{\text{EMD}} - \text{EMD}(\mu, \nu)\right| \leqslant \left|\widehat{\text{EMD}} - \text{EMD}(\hat{\mu}, \hat{\nu})\right| + \text{EMD}(\mu, \hat{\mu}) + \text{EMD}(\nu, \hat{\nu}) \leqslant 4\xi + 5\gamma.$$

Scaling γ down of a factor 5 we retireve Corollary 1.1. □

6.3 Proof of Theorem 2

Theorem 2 (Main theorem, graph interpretation). *For each constant $\gamma > 0$, there exists a constant $\varepsilon > 0$, and an algorithm running in time $O(n^{2-\varepsilon})$ with the following guarantees. The algorithm takes as input a budget B, and query access to the edge-cost matrix of an undirected, bipartite graph G over n vertices. The algorithm returns an estimate $\widehat{M}$ that is within $\pm\gamma n$ of the size of the maximum matching in G with total cost at most B.*

Proof. Let M be the maximum matching in G with total cost at most B, and let $|M| = \xi n$. We perform a binary search for ξ using the algorithm from Theorem 3 as a subroutine. This loses only a factor $\log(n)$ in query complexity, which gets absorbed in $O(n^{2-\varepsilon})$ by choosing a suitable constant ε. $\square$

Acknowledgments. We thank Erik Waingarten for inspiring discussions. We thank Tal Herman and Greg Valiant for pointing out the sample complexity lower bound implied by [VV10].

References

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein generative adversarial networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 214–223.

[Aga+22] Pankaj K Agarwal, Hsien-Chih Chang, Sharath Raghvendra, and Allen Xiao. “Deterministic, near-linear $(1+\epsilon)$-approximation algorithm for geometric bipartite matching”. In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 2022, pp. 1052–1065.

[AIK08] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. “Earth mover distance over high-dimensional spaces.” In: *SODA*. Vol. 8. 2008, pp. 343–352.

[ALT21] Tenindra Abeywickrama, Victor Liang, and Kian-Lee Tan. “Optimizing bipartite matching in real-world applications by incremental cost computation”. In: *Proceedings of the VLDB Endowment* 14.7 (2021), pp. 1150–1158.

[And+09] Alexandr Andoni, Khanh Do Ba, Piotr Indyk, and David Woodruff. “Efficient sketches for earth-mover distance, with applications”. In: *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE. 2009, pp. 324–330.

[And+14] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. “Parallel algorithms for geometric graph problems”. In: *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. 2014, pp. 574–583.

[ANR17] Jason Altschuler, Jonathan Niles-Weed, and Philippe Rigollet. “Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration”. In: *Advances in neural information processing systems* 30 (2017).

[AS14] Pankaj K Agarwal and R Sharathkumar. “Approximation algorithms for bipartite matching with metric and geometric costs”. In: *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. 2014, pp. 555–564.

[AZ23] Alexandr Andoni and Hengjie Zhang. “Sub-quadratic (1+\eps)-approximate Euclidean Spanners, with Applications”. In: *arXiv preprint arXiv:2310.05315* (2023).

[Ba+11] Khanh Do Ba, Huy L Nguyen, Huy N Nguyen, and Ronitt Rubinfeld. "Sublinear time algorithms for earth mover's distance". In: *Theory of Computing Systems* 48 (2011), pp. 428–442.

[Bac+20] Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. "Scalable nearest neighbor search for optimal transport". In: *International Conference on machine learning.* PMLR. 2020, pp. 497–506.

[Băd+05] Mihai Bădoiu, Artur Czumaj, Piotr Indyk, and Christian Sohler. "Facility location in sublinear time". In: *Automata, Languages and Programming: 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005. Proceedings 32.* Springer. 2005, pp. 866–877.

[Beh+23] Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. "Beating greedy matching in sublinear time". In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA).* SIAM. 2023, pp. 3900–3945.

[Beh22] Soheil Behnezhad. "Time-optimal sublinear algorithms for matching and vertex cover". In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS).* IEEE. 2022, pp. 873–884.

[BKS23a] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Dynamic $(1+\epsilon)$-Approximate Matching Size in Truly Sublinear Update Time". In: *arXiv preprint arXiv:2302.05030* (2023).

[BKS23b] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Sublinear Algorithms for $(1.5+\epsilon)$-Approximate Matching". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023.* Ed. by Barna Saha and Rocco A. Servedio. ACM, 2023, pp. 254–266. DOI: `10.1145/3564246.3585252`. URL: `https://doi.org/10.1145/3564246.3585252`.

[Bla+18] Jose Blanchet, Arun Jambulapati, Carson Kent, and Aaron Sidford. "Towards optimal running times for optimal transport". In: *arXiv preprint arXiv:1810.07717* (2018).

[BRR23] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. "Sublinear time algorithms and complexity of approximate maximum matching". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing.* 2023, pp. 267–280.

[Can20] Clément L Canonne. "A survey on distribution testing: Your data is big. But is it blue?" In: *Theory of Computing* (2020), pp. 1–100.

[Cha02] Moses S Charikar. "Similarity estimation techniques from rounding algorithms". In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing.* 2002, pp. 380–388.

[Che+22a] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum flow and minimum-cost flow in almost-linear time". In: *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS).* IEEE. 2022, pp. 612–623.

[Che+22b] Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. "New streaming algorithms for high dimensional EMD and MST". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing.* 2022, pp. 222–233.

[CS09] Artur Czumaj and Christian Sohler. “Estimating the weight of metric minimum spanning trees in sublinear time”. In: *SIAM Journal on Computing* 39.3 (2009), pp. 904–922.

[Cut13] Marco Cuturi. “Sinkhorn distances: Lightspeed computation of optimal transport”. In: *Advances in neural information processing systems* 26 (2013).

[DGK18] Pavel Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. “Computational optimal transport: Complexity by accelerated gradient descent is better than by Sinkhorn’s algorithm”. In: *International conference on machine learning*. PMLR. 2018, pp. 1367–1376.

[FL22] Kyle Fox and Jiashuai Lu. “A deterministic near-linear time approximation scheme for geometric transportation”. In: *arXiv preprint arXiv:2211.03891* (2022).

[GT89] Harold N Gabow and Robert E Tarjan. “Faster scaling algorithms for network problems”. In: *SIAM Journal on Computing* 18.5 (1989), pp. 1013–1036.

[HIS13] Sariel Har-Peled, Piotr Indyk, and Anastasios Sidiropoulos. “Euclidean spanners in high dimensions”. In: *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2013, pp. 804–809.

[Ind03] Piotr Indyk. “Fast color image retrieval via embeddings”. In: *Workshop on Statistical and Computational Theories of Vision (at ICCV), 2003*. 2003.

[Ind04] Piotr Indyk. “Algorithms for dynamic geometric problems over data streams”. In: *Proceedings of the thirty-sixth annual ACM Symposium on Theory of Computing*. 2004, pp. 373–380.

[Kuh55] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[Kus+15] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. “From word embeddings to document distances”. In: *International conference on machine learning*. PMLR. 2015, pp. 957–966.

[Le+21] Khang Le, Huy Nguyen, Quang M Nguyen, Tung Pham, Hung Bui, and Nhat Ho. “On robust optimal transport: Computational complexity and barycenter computation”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 21947–21959.

[LMR19] Nathaniel Lahn, Deepika Mulchandani, and Sharath Raghvendra. “A graph theoretic additive approximation of optimal transport”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[LXH23] Yiling Luo, Yiling Xie, and Xiaoming Huo. “Improved Rate of First Order Algorithms for Entropic Optimal Transport”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2023, pp. 2723–2750.

[McG05] Andrew McGregor. “Finding graph matchings in data streams”. In: *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer. 2005, pp. 170–181.

[PC19] Gabriel Peyré and Marco Cuturi. “Computational optimal transport: With applications to data science”. In: *Foundations and Trends® in Machine Learning* 11.5-6 (2019), pp. 355–607.

[Pha+20] Khiem Pham, Khang Le, Nhat Ho, Tung Pham, and Hung Bui. "On unbalanced optimal transport: An analysis of sinkhorn algorithm". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7673–7682.

[Roh19] Dhruv Rohatgi. "Conditional hardness of earth mover distance". In: *arXiv preprint arXiv:1909.11068* (2019).

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. "The earth mover's distance as a metric for image retrieval". In: *International journal of computer vision* 40 (2000), pp. 99–121.

[SA12] R Sharathkumar and Pankaj K Agarwal. "A near-linear time ε-approximation algorithm for geometric bipartite matching". In: *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. 2012, pp. 385–394.

[San15] Filippo Santambrogio. "Optimal transport for applied mathematicians". In: *Birkäuser, NY* 55.58-63 (2015), p. 94.

[Sol+15] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. "Convolutional wasserstein distances: Efficient optimal transportation on geometric domains". In: *ACM Transactions on Graphics (ToG)* 34.4 (2015), pp. 1–11.

[Vil+09] Cédric Villani et al. *Optimal transport: old and new*. Vol. 338. Springer, 2009.

[VV10] Gregory Valiant and Paul Valiant. "A CLT and tight lower bounds for estimating entropy." In: *Electron. Colloquium Comput. Complex.* Vol. 17. 2010, p. 179.

[Yur+19] Mikhail Yurochkin, Sebastian Claici, Edward Chien, Farzaneh Mirzazadeh, and Justin M Solomon. "Hierarchical optimal transport for document representation". In: *Advances in neural information processing systems* 32 (2019).

Appendix B

SODA: Better Sum Estimation via Weighted Sampling

Better Sum Estimation via Weighted Sampling

Lorenzo Beretta [*]
beretta@di.ku.dk
BARC, Univ. of Copenhagen

Jakub Tětek [*]
j.tetek@gmail.com
BARC, Univ. of Copenhagen

Abstract

Given a large set U where each item $a \in U$ has weight $w(a)$, we want to estimate the total weight $W = \sum_{a \in U} w(a)$ to within factor of $1 \pm \varepsilon$ with some constant probability $> 1/2$. Since $n = |U|$ is large, we want to do this without looking at the entire set U. In the traditional setting in which we are allowed to sample elements from U uniformly, sampling $\Omega(n)$ items is necessary to provide any non-trivial guarantee on the estimate. Therefore, we investigate this problem in different settings: in the *proportional* setting we can sample items with probabilities proportional to their weights, and in the *hybrid* setting we can sample both proportionally and uniformly. These settings have applications, for example, in sublinear-time algorithms and distribution testing.

Sum estimation in the proportional and hybrid setting has been considered before by Motwani, Panigrahy, and Xu [ICALP, 2007]. In their paper, they give both upper and lower bounds in terms of n. Their bounds are near-matching in terms of n, but not in terms of ε. In this paper, we improve both their upper and lower bounds. Our bounds are matching up to constant factors in both settings, in terms of both n and ε. No lower bounds with dependency on ε were known previously. In the proportional setting, we improve their $\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$ algorithm to $O(\sqrt{n}/\varepsilon)$. In the hybrid setting, we improve $\tilde{O}(\sqrt[3]{n}/\varepsilon^{9/2})$ to $O(\sqrt[3]{n}/\varepsilon^{4/3})$. Our algorithms are also significantly simpler and do not have large constant factors.

We then investigate the previously unexplored scenario in which n is not known to the algorithm. In this case, we obtain a $O(\sqrt{n}/\varepsilon + \log n/\varepsilon^2)$ algorithm for the proportional setting, and a $O(\sqrt{n}/\varepsilon)$ algorithm for the hybrid setting. This means that in the proportional setting, we may remove the need for advice without greatly increasing the complexity of the problem, while there is a major difference in the hybrid setting. We prove that this difference in the hybrid setting is necessary, by showing a matching lower bound.

Our algorithms have applications in the area of sublinear-time graph algorithms. Consider a large graph $G = (V, E)$ and the task of $(1 \pm \varepsilon)$-approximating $|E|$. We consider the (standard) settings where we can sample uniformly from E or from both E and V. This relates to sum estimation as follows: we set $U = V$ and the weights to be equal to the degrees. Uniform sampling then corresponds to sampling vertices uniformly. Proportional sampling can be simulated by taking a random edge and picking one of its endpoints at random. If we can only sample uniformly from E, then our results immediately give a $O(\sqrt{|V|}/\varepsilon)$ algorithm. When we may sample both from E and V, our results imply an algorithm with complexity $O(\sqrt[3]{|V|}/\varepsilon^{4/3})$. Surprisingly, one of our subroutines provides an $(1 \pm \varepsilon)$-approximation of $|E|$ using $\tilde{O}(d/\varepsilon^2)$ expected samples, where d is the average degree, under the mild assumption that at least a constant fraction of vertices are non-isolated. This subroutine works in the setting where we can sample uniformly from both V and E. We find this remarkable since it is $O(1/\varepsilon^2)$ for sparse graphs.

[*] Lorenzo Beretta and Jakub Tětek belong to Basic Algorithms Research Copenhagen (BARC), University of Copenhagen. BARC is supported by the VILLUM Foundation grant 16582. Jakub Tětek received funding from the Bakala Foundation. Lorenzo Beretta receives funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 801199.

1 Introduction.

Suppose we have a large set U, a weight function $w : U \to [0, \infty)$ and we want to compute a $(1 \pm \varepsilon)$-approximation of the sum of all weights $W = \sum_{a \in U} w(a)$. Since $n = |U|$ is very large, we want to estimate W by sampling as few elements as possible. In the traditional setting in which we are allowed to sample elements from U uniformly, sampling $o(n)$ items cannot provide any non-trivial guarantee on the approximation as we may miss an element with a very large weight. This led Motwani, Panigrahy, and Xu [12] to study this problem when we are allowed to sample elements proportionally to their weights (i.e., sample a with probability $w(a)/W$). In particular, they studied two settings: the *proportional* setting, where we can sample items proportionally to their weights, and the *hybrid* setting where both proportional and uniform sampling is possible. In this paper, we revisit these two settings and get both improved lower and upper bounds. We also extend the results to more general settings and show how our techniques imply new results for counting edges in sublinear time.

Motwani et al. [12] give upper and lower bounds for both proportional and hybrid settings. Their bounds are matching up to polylogarithmic factors in terms of n, but not in terms of ε. In this paper, we improve both their upper and lower bounds. Our bounds are matching up to *constant* factors in both settings, in terms of both n and ε. No lower bounds with dependency on ε were known previously.

In the proportional setting, we improve their $\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$ algorithm to $O(\sqrt{n}/\varepsilon)$. In the hybrid setting, we improve $\tilde{O}(\sqrt[3]{n}/\varepsilon^{9/2})$ to $O(\sqrt[3]{n}/\varepsilon^{4/3})$. Our algorithms are also significantly simpler. In the same paper, Motwani *et al.* [12] write: "to efficiently derive the sum from [proportional] samples does not seem straightforward". We would like to disagree and give a formula that outputs an estimate of W from a proportional sample. This formula is not only optimal in terms of sample complexity but also very simple. Our other algorithms, although not as simple, do not have large hidden constants and we believe they are both practical and less involved than their predecessors.

In their work, Motwani *et al.* [12] always assume to know the size of the universe $n = |U|$. We extend these results to the case of unknown n. In this case, we obtain a $O(\sqrt{n}/\varepsilon + \log n/\epsilon^2)$ algorithm for the proportional setting, and a $O(\sqrt{n}/\varepsilon)$ algorithm for the hybrid setting. This means that in the proportional setting we may remove the need for advice without significantly impacting the complexity of the problem, while there is a major difference in the hybrid setting. We prove that this difference in the hybrid setting is necessary, by showing a matching lower bound.

We give lower bounds for all our estimation problems, both for proportional and hybrid settings, and when n is either known or unknown. This is the most technically challenging part of the paper. We prove lower bounds for n known and unknown as well as proportional and hybrid settings; all our lower and upper bounds are matching up to a constant factor. See Table 1 for a summary of our results.

Our algorithms have particularly interesting applications in the area of sublinear-time graph algorithms, which are explained in detail in Section 1.1.

The paper is structured as follows. In what is left of Section 1 we explain applications and related work, give an overview of the techniques employed, and provide the reader with formal definitions of problems and notations. Section 2 contains our algorithms for proportional setting, while Section 3 contains algorithms for hybrid setting. In Section 4 we prove all our lower bounds. In Section 5 we show how to apply our algorithms to the problem of counting the number of edges in a graph in sublinear time. In Section 6 we raise several open problems.

Advice to the algorithm	This paper		Motwani, Panigrahy and Xu [12]	
	Proportional	Hybrid	Proportional	Hybrid
n known	$\Theta(\sqrt{n}/\varepsilon)$	$O(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n\log n))$ $\Omega(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n)$	$\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$ $\Omega(\sqrt{n})$	$\tilde{O}(\sqrt[3]{n}/\varepsilon^{9/2})$, $O(\sqrt{n}/\varepsilon^2)$ $\Omega(\sqrt[3]{n})$
Known $\tilde{n} \geq n$	$O(\sqrt{\tilde{n}}/\varepsilon)$ $\Omega(\sqrt{n}/\varepsilon)$	$O(\min(\sqrt{n}/\varepsilon, n\log n))$ $\Omega(\min(\sqrt{n}/\varepsilon, n))$		
No advice	$O(\sqrt{n}/\varepsilon + \log n/\varepsilon^2)$ $\Omega(\sqrt{n}/\varepsilon)$	$O(\min(\sqrt{n}/\varepsilon, n\log n))$ $\Omega(\min(\sqrt{n}/\varepsilon, n))$		

Table 1: Results of this paper.

1.1 Related Work and Applications.

In this section, we show that our algorithms can be applied to get sublinear-time graph algorithms and distribution testing and discuss how this relates to previous work in these areas. We also discuss here how the widely used Metropolis-Hastings algorithm in fact implements the proportional sampling. We suggest that this could be an application domain worth investigating.

Counting edges in sublinear time.

When a graph $G = (V, E)$ is very large, we may want to approximately solve certain tasks without looking at the entire G, thus having a time complexity that is sublinear in the size of G. In particular, estimating global properties of G such $|V|$ or $|E|$ in this setting is an important problem and has been studied in both theoretical and applied communities [4, 5, 6, 8, 10, 14, 16]. Since the algorithm does not have the time to pre-process (or even see) the whole graph, it is important to specify how we access G. Several models are employed in the literature. The models differ from each other for the set of queries that the algorithm is allowed to perform. A *random vertex query* returns a random vertex, a *random edge query* returns a random edge, a *pair query* takes two vertices u, v as arguments and returns $(u, v) \in E$, a *neighbourhood query* takes a vertex v and an index i as arguments and returns the i-th neighbour of v (or says that $d(v) < i$), a *degree query*, given a vertex v, returns its degree $deg(v)$. We parameterize the complexities with an approximation parameter ε, $n = |V|$ and $m = |E|$.

The problem of estimating the number of edges in a graph in sublinear time has been first considered by Feige [6]. Their algorithm works in the model where only random vertex queries and degree queries are allowed and achieves a $(2 + \varepsilon)$-approximation algorithm using $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}})$ time and queries. Their algorithm does not use neighborhood queries and the authors showed that without neighborhood queries, $2 - \varepsilon$ approximation requires a linear number of queries. Goldreich and Ron [8] broke the barrier of factor 2 by using neighborhood queries. Indeed, they showed a $(1 + \varepsilon)$-approximation with time and query complexity of $\tilde{O}(\frac{n}{\varepsilon^{9/2}\sqrt{m}})$. Currently, the best known algorithm is the one by Eden, Ron and Seshadhri [5] and has complexity $\tilde{O}(\frac{n}{\varepsilon^2\sqrt{m}})$. If pair queries are allowed, the algorithm of Tětek and Thorup [16] has complexity $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^4})$; in the same paper the authors showed a lower bound which is near-matching for $\varepsilon \geq m^{1/6}/n^{1/3}$ as well as an algorithm with complexity $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^2})$ in what they call the hash-ordered access model. They also show an algorithm in the more standard setting with random vertex and neighborhood queries, that runs in time $\tilde{O}(\sqrt{n}/\varepsilon)$. This is the same complexity (up to a $\log^{O(1)} n$ factor) that we achieve in the setting with random edge queries, as we discuss below. Our techniques are, however, completely different and share no similarity with the techniques used in [16].

Our algorithms can be applied to solve the edge counting problem when either (i) random edge queries only are allowed, or (ii) both random edge and random vertex queries are allowed. While edge counting has not been explicitly considered in these settings before, these settings are established and have been used in several papers [1, 2, 7, 15]. We instantiate our algorithm for this graph problem setting $U = V$ and $w(v) = deg(v)$ for each $v \in V$. Uniform sampling then corresponds to sampling vertices uniformly. Proportional sampling can be simulated by taking a random edge and picking one of its endpoints at random. Degree query allows us to get the weights of sampled vertices. Since these settings have not been explicitly studied before, we compare our results with what follows directly from the known literature.

If we can only sample uniformly from E, the algorithm by Motwani et al. implies an algorithm for this problem that has complexity $\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$. Using an algorithm from [5] and standard simulation of random vertex queries using random edge queries, one would get time $\tilde{O}(\sqrt{m}/\varepsilon^2 + \frac{m}{\varepsilon\sqrt{n'}})$ for n' being the number of non-isolated[1] vertices[2]. Our results immediately give a $O(\sqrt{n}/\varepsilon)$ algorithm, or $\tilde{O}(\sqrt{n}/\varepsilon + 1/\varepsilon^2)$ when n is not known.

When we may sample both from E and V, the algorithms by Motwani et al. imply algorithms with sample complexities of $\tilde{O}(\sqrt[3]{n}/\varepsilon^{9/2})$ and $O(\sqrt{n}/\varepsilon^2)$. We may also use the algorithm of [5] that relies on random vertex query only. This has complexity from $O(\frac{n}{\varepsilon^2\sqrt{m}})$. Our results imply an algorithm with complexity $O(\sqrt[3]{n}/\varepsilon^{4/3})$. Surprisingly, one of our subroutines provides an $(1 \pm \varepsilon)$-approximation of $|E|$ using $\tilde{O}(d/\varepsilon^2)$ expected samples, where d is the average degree, under the mild assumption that at least a constant fraction of vertices are non-isolated (in fact, we prove a more complicated complexity which depends on the fraction of vertices that are isolated). This subroutine works in the setting where we can sample uniformly from both V and E. We find this remarkable since it is $O(1/\varepsilon^2)$ for sparse graphs.

Distribution testing.

Consider a model in which we are allowed to sample from a distribution $\mathcal{D}$ on U, and when we do, we receive both $a \in U$ and $P_{\mathcal{D}}(a)$. This model is stronger than the one we considered throughout the paper. In fact, it is a special case of our model obtained by fixing $W = 1$. Similarly, we may consider such stronger variant of the hybrid setting. This model has received attention both in statistics and in theoretical computer science literature. Most notably, Horvitz and Thompson [9] in 1952 showed how to estimate the sum $\sum_{a \in U} v(a)$, for value function $v : U \to \mathbb{R}$ when having access to a sample a from $\mathcal{D}$ and $P_{\mathcal{D}}(a)$. More recently, many distribution testing problems have been considered in this and similar models.

First, Canonne and Rubinfeld [3] considered the model where one can (i) sample $a \in U$ according to $\mathcal{D}$; (ii) query an oracle that, given $a \in U$, returns $P_{\mathcal{D}}(a)$. This allows us to both get the probability of a sampled item, but also the probability of any other item. For every permutation-invariant problem, any sublinear-time algorithm in this model may be transformed so that it only queries the oracle on previously sampled items or on items chosen uniformly at random [3]. This

[1] A vertex is isolated if it has degree zero.

[2] One may simulate uniform sampling from the set of non-isolated vertices at multiplicative overhead of $O(m/n')$ by sampling proportionally and using rejection sampling. We may then use set size estimation by bithday paradox in time $O(\sqrt{n'}/\epsilon)$ to learn n' and the algorithm of [5].

[3] Since we consider permutation invariant problems, we may randomly permute the elements. Whenever we query an element that has not yet been sampled, we may assume that it is sampled uniformly from the set of not-yet-sampled elements. This can be simulated by sampling elements until getting a not-yet-seen item (sublinearity ensures that the step adds multiplicative $O(1)$ overhead).

setting is stronger than our hybrid setting. However, the two models become equivalent if we know W.

Second, Onak and Sun [13] considered exactly the model described above, where one can sample from a distribution $\mathcal{D}$, and both $a \in U$ and $P_{\mathcal{D}}(a)$ are returned. This setting is again stronger than our proportional setting, and it becomes equivalent once we know W.

In both of these papers, the authors solved several distribution testing problems such as: uniformity testing, identity testing, and distance to a known distribution, in the respective models. Canonne and Rubinfeld [3] also considered the problem of providing an additive approximation of entropy. Both [3] and [13] proved several lower bounds, showing that many of their algorithms are optimal up to constant factors. These lower bounds also imply a separation between the two models for the problems mentioned above.

In both of the papers, the authors make a point that many of their algorithms (all the ones we mentioned) are robust with respect to noise of multiplicative error $(1 \pm \varepsilon/2)$ in the answers of the probability oracles. Our algorithms can then serve as a reduction from the weaker models we consider in this paper to these stronger models where we know the probabilities and not just the weights. The reason is that we can get an approximation of W, meaning that we may then approximately implement the above-described models. Since the mentioned algorithms [3, 13] are robust, a $(1 \pm \varepsilon/2)$-approximation of W is sufficient to simulate them.

Proportional sampling in practice: Metropolis-Hastings.

Proportional sampling is often implemented in practice using the Metropolis-Hasting algorithm. This algorithm is widely used in statistics and statistical physics, but can also be used to sample combinatorial objects. It can be used to sample from large sets which have complicated structures that make it difficult to use other sampling methods. Just like our algorithms, it is suitable when the set is too large to be stored explicitly, making it impossible to pre-process it for efficient sampling. One of the main appeals of Metropolis-Hasting is that it does not require one to know the exact sampling probabilities, but it is sufficient to know the items' weights like in the proportional setting described in this paper. We believe that our algorithm can find practical applications in combination with Metropolis-Hasting as Metropolis-Hasting performs proportional sampling with weight function that would be usually known in practice.

1.2 Overview of employed techniques.

Here we provide a summary of the techniques employed throughout the paper. We denote with $P_{unif}(\cdot)$ and $P_{prop}(\cdot)$ the probabilities computed according to uniform and proportional sampling, respectively. Although often not specified for brevity, all guarantees on the approximation factors of estimates in this section are meant to hold with probability 2/3.

1.2.1 Proportional setting with advice $\tilde{n} \geq n$.

Consider sampling two elements $a_1, a_2 \in U$ proportionally and define $Y_{12} = 1/w(a_1)$ if $a_1 = a_2$, and $Y_{12} = 0$ otherwise. It is easy to show that Y_{12} is an unbiased estimator of $1/W$. We could perform this experiment many times and take the average. This would give a good approximation to $1/W$ and taking the inverse value, we would get a good estimate on W. Unfortunately, we would need $\Theta(n/\varepsilon^2)$ repetitions in order to succeed with a constant probability. We can fix this as follows: we take m samples $a_1, \cdots, a_m$ and consider one estimator Y_{ij} for each pair of samples a_i, a_j for $i \neq j$. This allows us to get $\binom{m}{2}$ estimators from m samples. We show that estimators Y_{ij} are uncorrelated. This reduces the needed number of samples from $O(n/\varepsilon^2)$ to $O(\sqrt{n}/\varepsilon)$.

We now describe the estimator formally. Let $S = \{a_1, \cdots, a_m\}$ be the set of sampled items, and for each $s \in S$ define c_s to be the number of times item s is sampled. Then,

$$\hat{W} = \binom{m}{2} \cdot \left(\sum_{s \in S} \frac{\binom{c_s}{2}}{w(s)} \right)^{-1}$$

is a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$ for $m = \Theta(\sqrt{n}/\varepsilon)$. If, instead of knowing n exactly, we know some $\tilde{n} \geq n$ we can achieve the same guarantees by taking $\Theta(\sqrt{\tilde{n}}/\varepsilon)$ samples.

1.2.2 Proportional setting, unknown n.

We partition U into buckets $B_i = \{a \in U \mid w(a) \in [2^i, 2^{i+1})\}$. We choose some $b \in \mathbb{Z}$ and compute two estimates: $\hat{W}_b$ approximates $W_b = \sum_{a \in B_b} w(a)$, and $\hat{P}_b$ approximates $P_b = P_{prop}(a \in B_b) = W_b/W$. We then return $\hat{W}_b/\hat{P}_b$, as an estimate of W. If both estimates are accurate, then the returned value is accurate. To choose b, we sample a_1 and a_2 proportionally and define b so that $\max\{a_1, a_2\} \in [2^b, 2^{b+1})$. We prove that, surprisingly[4], $E[1/P_b] = O(\log n)$. Computing $\hat{P}_b$ is simply a matter of estimating the fraction of proportional samples falling into B_b. This can be done using $O(1/(P_b\varepsilon^2))$ samples, where ε is the approximation parameter. Therefore the expected complexity of computing $\hat{P}_b$ is $O(\log n/\varepsilon^2)$. Computing $\hat{W}_b$ is more involved. First, we design two subroutines to sample from B_b, one for proportional sampling and one for uniform. These subroutines work by sampling $a \in U$ until we get $a \in B_b$; the subroutine for uniform sampling then uses rejection sampling. These subroutines take, in expectation, $O(1/P_b)$ samples to output one sample from B_b. Since we can now sample uniformly from B_b, we sample items uniformly form B_b until we find the first repeated item and use the stopping time to infer $|B_b|$ up to a constant factor. In expectation, we use $O(\sqrt{|B_b|})$ uniform samples and compute $\tilde{n}_b$, such that $|B_b| \leq \tilde{n}_b \leq O(|B_b|)$. Since we can also sample proportionally from B_b, we may use the algorithm for proportional setting with advice $\tilde{n}_b$ to estimate W_b. This yields a total sample complexity of $O(\sqrt{n}/\varepsilon + \log n/\varepsilon^2)$.

1.2.3 Hybrid setting.

We now present the techniques used in the hybrid setting. Before giving a sketch of the main algorithms, we introduce two subroutines.

Coupon-collector-based algorithm. In the hybrid setting, we can sample elements uniformly. If we know $n = |U|$ a well-known result under the name of "coupon collector problem" shows that we can retrieve with high probability all n elements by performing $\Theta(n \log n)$ uniform samples. We extend this result to the case of unknown n. We maintain a set of retrieved elements $S \subseteq U$ and keep on sampling uniformly and adding new elements to S until we perform $\Theta(|S| \log |S|)$ samples in a row without updating S. It turns out that this procedure retrieves the whole set U with probability $2/3$ and expected complexity $O(n \log n)$. According to our lower bounds, this algorithm is near-optimal when $\varepsilon = O(1/\sqrt{n})$. Therefore, we can focus on studying hybrid sampling for larger values of ε.

Harmonic mean and estimating W/n with advice $\tilde{\theta} \geq W/n$ in hybrid setting. If we sample $a \in U$ proportionally, then $1/w(a)$ is an unbiased estimator of n/W. Unfortunately, we may have very small values of $w(a)$, which can make the variance of this estimator arbitrarily large.

[4]Notice that, if we just define $b = a_1$, then we have $E[1/P_b] = n$ in the worst case. Therefore, taking the maximum of two samples entails an exponential advantage.

To reduce variance, we can set a threshold ϕ and define an estimator Y_ϕ as $1/w(a)$ if $w(a) \geq \phi$, and 0 otherwise. Set $p = P_{unif}(w(a) \geq \phi)$, then we have $E[Y_\phi] = pn/W$ and $Var(Y_\phi) \leq pn/(\phi W)$. If we define $\bar{Y}_\phi$ as the average of $\tilde{\theta}/(p\phi\varepsilon^2)$ copies of Y_ϕ, we have $Var(\bar{Y}_\phi) \leq (\varepsilon E[Y_\phi])^2$. With such a small variance, $\bar{Y}_\phi$ is a good estimate of pn/W. Estimating p is simply a matter of estimating the fraction of uniform samples having $w(\bullet) \geq \phi$. This can be done taking $O(1/(p\varepsilon^2))$ uniform samples. Once we have estimates of both p and pn/W we return their ratio as an estimate of W/n. We employed in total $O((1 + \frac{\tilde{\theta}}{\phi})/(p\,\varepsilon^2))$ proportional and uniform samples.

Perhaps surprisingly, this corresponds to taking the harmonic mean of the samples with weight at least ϕ and adjusting this estimate for the weight of the items with weight $< \phi$.

Hybrid setting, known n. We now sketch the algorithm for sum estimation in the hybrid setting with known n. We combine our formula to estimate W in the proportional setting and the above algorithm to estimate W/n in hybrid setting to obtain an algorithm that estimates W using $O(\sqrt[3]{n}/\varepsilon^{4/3})$ uniform and proportional samples. Define $U_{\geq\theta} = \{a \in U \mid w(a) \geq \theta\}$. We find a θ such that $P_{unif}(a \in U_{\geq\theta}) \approx n^{-1/3}\varepsilon^{-2/3}$, this can be done taking enough uniform samples and picking the empirical $(1 - n^{-1/3}\varepsilon^{-2/3})$-quantile[5] . We define $p = P_{prop}(a \in U_{\geq\theta}) \geq P_{unif}(a \in U_{\geq\theta}) \approx n^{-1/3}\varepsilon^{-2/3}$. We compute an estimate $\hat{p}$ of p counting the fraction of proportional samples falling in $U_{\geq\theta}$. Now we have two cases. If $\hat{p} \geq 1/2$, we can simulate sampling from the proportional distribution on $U_{\geq\theta}$ with $O(1)$ overhead by sampling proportionally until we get an element of $U_{\geq\theta}$. Then, we use the algorithm for sum estimation under proportional sampling restricted to elements in $U_{\geq\theta}$ to estimate $\sum_{a\in U_{\geq\theta}} w(a) = pW$. Dividing by the estimate $\hat{p} \approx p$, we get an estimate for W. Else, $\hat{p} < 1/2$, and thus $p \leq 2/3$ (assuming $\hat{p}$ is a good enough estimate of p). We then have $\theta n \geq \sum_{a\notin U_{\geq\theta}} w(a) = (1-p)W \geq W/3$. This allows us to use the harmonic-mean-based algorithm to estimate W/n with $\phi = \theta$ and $\tilde{\theta} = 3\theta$. In both cases we manage to provide an estimate of W using $O(\sqrt[3]{n}/\varepsilon^{4/3})$ samples.

Hybrid setting, unknown n. We now sketch our technique for sum estimation in the hybrid setting with unknown n. First, we sample items uniformly until we find the first repeated item and use the stopping time to infer the total number of items n. In expectation, we use $O(\sqrt{n})$ uniform samples and compute $\tilde{n}$, such that $n \leq \tilde{n}$ and $E[\tilde{n}] = n$. Now, we can use our algorithm for sum estimation in the proportional setting with advice $\tilde{n}$. This uses in expectation $O(\sqrt{\tilde{n}}/\varepsilon) = O(\sqrt{n}/\varepsilon)$ samples. As we prove, this complexity is optimal up to a constant factor for $\epsilon \geq 1/\sqrt{n}$. Again, if $\varepsilon \leq 1/\sqrt{n}$, then we use our coupon-collector-based algorithm that retrieves every element of U using $O(n \log n)$ samples.

1.2.4 Lower bounds.

The most technically challenging part of our paper is Section 4, where we prove lower bounds for all estimation problems we address in the first part of the paper. All our lower bound proofs follow a common thread. We now sketch the main ideas. First, we define two different instances of the estimation problem at hand (U_1, w_1) and (U_2, w_2) such that a $(1 \pm \varepsilon)$-approximation of W is sufficient to distinguish between them. Then, we define our *hard* instance as a mixture of the two: we take (U_1, w_1) with probability 1/2 and (U_2, w_2) otherwise. We denote these events by $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively. Second, we show that a Bayes classifier cannot distinguish between the two cases

[5]We may assume that $\epsilon \geq 8/\sqrt{n}$ by running the coupon-collector-based algorithm when $\epsilon < 8/\sqrt{n}$. Under this assumption, it holds $n^{-1/3}\varepsilon^{-2/3} \in [0, 1]$ and taking the $(1 - n^{-1/3}\varepsilon^{-2/3})$-quantile then is meaningful.

with probability 2/3 using too few samples; since Bayes classifiers are risk-optimal[6] this implies that no classifier can have misclassification probability less than 1/3 while using the same number of samples. To show that a Bayes classifier has a certain misclassification probability, we study the posterior distribution, conditioned on the samples it has seen. Let the multiset S represent the outcome of the samples. Since the prior is uniform, applying the Bayes theorem gives

$$\frac{P(\mathcal{E}_1 \mid S)}{P(\mathcal{E}_2 \mid S)} = \frac{P(S \mid \mathcal{E}_1)}{P(S \mid \mathcal{E}_2)}.$$

We denote this likelihood ratio by $\mathcal{R}(S)$. We show that whenever $|S|$ is (asymptotically) too small, we have $\mathcal{R}(S) \approx 1$ with probability close to 1. When $\mathcal{R}(S) \approx 1$, the posterior distribution is very close to uniform. This entails misclassification probability close to 1/2. If X and Y are some random variables sufficient to reconstruct S (that is, there exists an algorithm that, given (X, Y), generates S' with the same distribution as S), we can define

$$\mathcal{R}(X) = \frac{P(X \mid \mathcal{E}_1)}{P(X \mid \mathcal{E}_2)} \quad \text{and} \quad \mathcal{R}(Y \mid X) = \frac{P(Y \mid X, \mathcal{E}_1)}{P(Y \mid X, \mathcal{E}_2)}$$

and we have, thanks to the Bayes theorem, $\mathcal{R}(S) = \mathcal{R}(X) \cdot \mathcal{R}(Y \mid X)$. In this way, we can break the problem of proving $\mathcal{R}(S) \approx 1$ into proving $\mathcal{R}(X) \approx 1$ and $\mathcal{R}(Y \mid X) \approx 1$. This allows us to reduce all our lower bounds to prove concentration of likelihood ratios for two basic problems: (1) distinguishing between two sets of size n and $(1-\varepsilon)n$ by uniform sampling and (2) distinguishing between two sequences of i.i.d. random variables from $Bern(p)$ and $Bern(p-\varepsilon)$.

We now sketch how we bound the likelihood ratio $\mathcal{R}(S)$ for problem (1), the same technique applies to problem (2). In problem (1), we call $\mathcal{E}_1$ the event $|U| = n$ and $\mathcal{E}_2$ the event $|U| = (1-\varepsilon)n$, and we set $P(\mathcal{E}_1) = P(\mathcal{E}_2) = 1/2$. First, we notice that $\mathcal{R}(S)$ depends only on the number $\ell(S)$ of distinct elements in S. Then, we prove three facts. First, $\ell(S) \mid \mathcal{E}_i$ is concentrated around $E[\ell(S) \mid \mathcal{E}_i]$. Second, $E[\ell(S) \mid \mathcal{E}_1] \approx E[\ell(S) \mid \mathcal{E}_2]$. Third, for small deviations of $\ell(S)$, we have small deviations of $\mathcal{R}(\ell(S))$. These three facts are sufficient to conclude that, with probability close to 1, $R(\ell(S))$ lies in a very narrow interval; further computations show that 1 lies in that interval, hence $\mathcal{R}(S) \approx 1$ with probability close to 1.

1.3 Preliminaries.

Problem definition. We now give a formal definition of the two settings that we consider. Let us have a set U of cardinality n and a weight function $w : U \to [0, \infty)$. We denote by W the sum $\sum_{a \in U} w(a)$. The following operations are allowed in the proportional sampling setting: (1) proportionally sample an item, this returns $(a, w(a))$ with probability $w(a)/W$; (2) given two items a, a', check whether $a = a'$. This is the only way we can interact with the items. In the hybrid setting, we may in addition (3) sample an item uniformly (that is, return $(a, w(a))$ for any $a \in U$ with probability $1/n$). In both settings, we want to compute an estimate $\hat{W}$ of W such that $(1-\varepsilon)W \le \hat{W} \le (1+\varepsilon)W$ with probability 2/3.

Notation. When $(1-\varepsilon)W \le \hat{W} \le (1+\varepsilon)W$ holds, we say that such $\hat{W}$ is a $(1 \pm \varepsilon)$-approximation of W. Some of our subroutines require "advice" in the form of a constant factor approximation of some value. For sake of consistency, we denote this constant factor approximation of $\bullet$ by $\tilde{\bullet}$. Similarly, if we want to estimate some value $\bullet$, we use $\hat{\bullet}$ to denote the estimate. Let us have some predicate ϕ that evaluates true on some subset of U and false on the rest. We denote by

[6]Risk of a classifier refers to the misclassification probability under some fixed distribution. For the Bayes classifier, we implicitly assume that this distribution is the same as the prior.

$P_{unif}(\phi(a))$ and $P_{prop}(\phi(a))$ the probability of ϕ evaluating to true for a being picked uniformly and proportionally, respectively.

We use $\tilde{O}$ with the slightly non-standard meaning of $f(n) \in \tilde{O}(g(n))$ being equivalent to $f(n) \in g(n) \log^{O(1)} n$, rather than $f(n) \in g(n) \log^{O(1)} g(n)$. We state all our results (both upper and lower bound) for some constant success probability $> 1/2$. These probabilities can be amplified to any other constants without increasing the asymptotic complexity. In pseudocode, we often say that we execute some algorithm with some failure probability. By this, we mean that one uses probability amplification to achieve that failure probability.

Relative bias estimation of a Bernoulli random variable. Let $X_1, X_2, \cdots$ be i.i.d. random variables distributed as $Bern(p)$. [11] gave a very simple algorithm that returns $\hat{p}$ such that, with probability at least 2/3, $\hat{p}$ is a $(1 \pm \varepsilon)$-approximation of p[7]. It can be summarized as follows.

Proposition 1 (follows from [11])**.** *Let $X_1, X_2, \cdots$ be i.i.d. random variables distributed as $Bern(p)$. There exists an algorithm that uses in expectation $O(\frac{1}{\varepsilon^2 p})$ samples and returns $\hat{p}$ such that $E[1/\hat{p}] = 1/p$ and*

$$P\left(|\hat{p} - p| > \varepsilon p\right) \leq \frac{1}{3}.$$

We call this algorithm BernoulliEstimator(ε). We assume that this algorithm has access to the sequence $X_1, X_2, \cdots$; we specify these random variables when invoking the algorithm.

Probability amplification and expected values. Consider an estimator that gives a guarantee on the estimate $\hat{x}$ that holds with some probability (say, guarantee that a proposition $\phi(\hat{x})$ holds with probability at least 2/3) and at the same time, we know that $E[\hat{x}] \leq y$ for some value y. We sometimes need to amplify the probability of the guarantee (that is, amplify the probability that $\phi(\hat{x})$ holds) but would like to retain a bound $E[\hat{x}] = O(y)$. We now argue that using the standard median trick is sufficient. Namely, we prove that

Lemma 2. *Let us have non-negative i.i.d. random variables $X_1 \cdots X_{2t-1}$ for some integer t, and let $X = median(X_1 \cdots X_{2t-1})$. It holds $E[X] \leq 2E[X_1]$.*

Proof. Let $X'_1, \cdots, X'_{2t-1}$ be the random variables $X_1, \cdots, X_{2t-1}$ sorted in increasing order. We then have

$$E[X] \leq E\left[\frac{1}{t} \cdot \sum_{i=t}^{2t-1} X'_i\right] \leq E\left[\frac{1}{t} \cdot \sum_{i=1}^{2t-1} X_i\right] \leq \frac{2t-1}{t} \cdot E[X_1].$$

□

2 Sum Estimation by Proportional Sampling.

In this section, we focus on sum estimation in the proportional setting. We design algorithms to estimate W and our objective is to minimize the total number of samples taken in the worst case. We present two different algorithms that provide an $(1 \pm \varepsilon)$-approximation of W with probability 2/3. The first one, PropEstimator, assumes to have an upper bound on the number of elements $\tilde{n} \geq n$, and achieves sample complexity of $O(\sqrt{\tilde{n}}/\varepsilon)$. The second one, NoAdvicePropEstimator, does not assume any knowledge of n, and produce an ε-estimate using $O(\sqrt{n}/\varepsilon + \log n/\varepsilon^2)$ samples in expectation.

[7]In that paper, the authors in fact solve a more general problem. For presentation of this special case, see [17].

2.1 Algorithm with advice $\tilde{n} \geq |U|$.

Let $a_1 \dots a_m$ be m items picked independently at random from U with probabilities proportional to their weights. Let S be the set of sampled items, and for each $s \in S$ define c_s to be the number of times item s is sampled. For each $i, j \in [m]^2$ define Y_{ij} to be $1/w(a_i)$ if $a_i = a_j$ and 0 otherwise. We now estimate W as follows:

Algorithm 1: PropEstimator$(\tilde{n}, \varepsilon)$**:**

Given a parameter $0 < \varepsilon < 1$ and advice $\tilde{n} \geq n$, perform $m = \sqrt{24\tilde{n}}/\varepsilon + 1$ samples and return the estimate

$$\hat{W} = \binom{m}{2} \cdot \left(\sum_{s \in S} \frac{\binom{c_s}{2}}{w(s)} \right)^{-1}.$$

In case $c_s = 1$ for all $s \in S$ set $\hat{W} = \infty$.

Before we prove correctness, we need the following lemma.

Lemma 3. *Given pairwise distinct $i, j, k \in [m]$, we have $E[Y_{i,j}] = 1/W$, $Var(Y_{i,j}) \leq n/W^2$, and $Cov[Y_{i,j}, Y_{i,k}] = 0$*

Proof.

$$E[Y_{ij}] = \sum_{a \in U} \frac{1}{w(a)} P(x_i = x_j = a) = \sum_{a \in U} \frac{w(a)}{W^2} = \frac{1}{W}$$

$$Var(Y_{ij}) \leq E[Y_{ij}^2] = \sum_{a \in U} \frac{1}{w(a)^2} P(x_i = x_j = a) = \sum_{a \in U} \frac{1}{W^2} = \frac{n}{W^2}$$

As for the covariance, it holds $Cov[Y_{i,j}, Y_{i,k}] = E[Y_{i,j} \cdot Y_{i,k}] - E[Y_{j,k}] \cdot E[Y_{i,k}]$. This is equal to 0 as

$$E[Y_{i,j} \cdot Y_{i,k}] = \sum_{a \in U} \frac{1}{w(a)^2} P(x_i = x_j = x_k = a) = \sum_{a \in U} \frac{w(a)}{W^3} = \frac{1}{W^2} = E[Y_{j,k}] \cdot E[Y_{i,k}]$$

□

Theorem 4. *Given parameters $\tilde{n}$ and $0 < \varepsilon < 1$,* PropEstimator$(\tilde{n}, \varepsilon)$ *has sample complexity $O(\frac{\sqrt{\tilde{n}}}{\varepsilon})$ and returns an estimate $\hat{W}$ such that $E[1/\hat{W}] = 1/W$. If, moreover, $\tilde{n} \geq n$, then $P(|\hat{W} - W| \leq \varepsilon W) \geq 2/3$.*

Proof. The sample complexity is clearly as claimed. We now prove that $1/\hat{W}$ is an unbiased estimator of $1/W$:

$$\frac{1}{\hat{W}} = \binom{m}{2}^{-1} \sum_{s \in S} \frac{\binom{c_s}{2}}{w(s)} = \binom{m}{2}^{-1} \sum_{1 \leq i < j \leq m} Y_{ij}$$

and thus

$$E\left[\frac{1}{\hat{W}}\right] = \binom{m}{2}^{-1} \sum_{1 \leq i < j \leq m} E\left[Y_{ij}\right] = \frac{1}{W}.$$

When i, j, k, ℓ are all distinct, Y_{ij} and $Y_{k\ell}$ are independent. Moreover, by Theorem 3, Y_{ij} and Y_{ik} are uncorrelated for $j \neq k$. Using the bound on $Var(Y_{ij})$ from Theorem 3, we then have that

$$\begin{aligned} Var\left[\frac{1}{\hat{W}}\right] &= \binom{m}{2}^{-2} \sum_{1 \le i < j \le m} Var(Y_{ij}) \\ &\le \binom{m}{2}^{-1} \frac{n}{W^2} \\ &\le \frac{1}{12} \cdot \left(\frac{\varepsilon}{W}\right)^2 \end{aligned}$$

By Chebyshev inequality, it holds that

$$P\left(\left|\frac{1}{\hat{W}} - \frac{1}{W}\right| > \frac{\varepsilon}{2W}\right) \le \frac{Var\left[1/\hat{W}\right]}{(\varepsilon/2W)^2} \le \frac{1}{3}$$

Finally, for $\varepsilon \le 1$ we have

$$(1-\varepsilon)W \le (1+\varepsilon/2)^{-1}W \le \hat{W} \le (1-\varepsilon/2)^{-1}W \le (1+\varepsilon)W$$

This means that $|1/\hat{W} - 1/W| \le \varepsilon/(2W)$ implies $|\hat{W} - W| \le \varepsilon W$. Thus $P\left(|\hat{W} - W| \le \varepsilon W\right) \ge 2/3$.
□

2.2 Algorithms for $|U|$ unknown.

In this section, we present the algorithm NoAdvicePropEstimator, which samples elements from U proportionally to their weights, and computes an $(1 \pm \varepsilon)$-approximation of W with probability $2/3$, without any knowledge of $n = |U|$.

NoAdvicePropEstimator takes $O(\sqrt{n}/\varepsilon + \log n/\epsilon^2)$ samples in expectation and works as follows. We partition U into buckets such that items in one bucket have roughly the same weight. We pick one bucket such that the items in this bucket are likely to have a sufficiently large total weight. We then estimate the sum restricted to this bucket. If we are able to do that, we can estimate the total weight by looking at what fraction of the proportional samples end up in this bucket. We estimate the sum restricted to the bucket as follows. Since the weights are roughly the same for all items in the bucket, we may use rejection sampling to efficiently simulate uniform samples from the bucket. That allows us to estimate the number of items in it, up to a constant factor. We use the algorithm PropEstimator with this estimated bucket size as the advice $\tilde{n}$.

Estimating $|U|$ through uniform sampling.

As a preliminary step, we assume that we are able to sample elements from U uniformly, rather than according to their weights. Under this assumption, we introduce the algorithm SetSizeEstimator that, using $O(\sqrt{n})$ expected samples, estimates $n = |U|$ up to a constant factor with probability $2/3$. The intuition behind SetSizeEstimator is fairly simple: if we sample uniformly with replacement from a universe of size n and we see the first repetition after t samples, then it is likely that $t \approx \sqrt{n}$.

Theorem 5. SetSizeEstimator *has expected sample complexity of* $O(\sqrt{n})$. *It returns an estimate* $\hat{N}$ *such that* $P(n \le \hat{N}) \ge 2/3$ *and* $E[\hat{N}] = O(n)$.

Algorithm 2: SetSizeEstimator()

```
1 $S_0 \leftarrow \emptyset$
2 for $i \in \mathbb{N}$ do
3     Sample $a_i \in U$ uniformly
4     if $a_i \in S_i$ then
5         $\hat{s} \leftarrow |S_i|$
6         $\hat{N} \leftarrow 4\hat{s}^2$
7         return $\hat{N}$
8     $S_{i+1} \leftarrow S_i \cup \{a_i\}$
```

Proof. We prove that when the algorithm aborts, it holds $P(\sqrt{n}/2 \le \hat{s}) \ge 2/3$. The bound on $P(n \le \hat{N})$ follows by the definition of $\hat{N}$. Define the event $\mathcal{E}_i = \{a_i \in S_i\}$, where we define that $a_i \notin S_i$ whenever the algorithm terminates before step i. It then holds $P(\mathcal{E}_i) \le P(\mathcal{E}_i \mid \bigcap_{j<i} \bar{\mathcal{E}}_j) = i/n$. We have

$$\begin{aligned} P\left(\hat{s} < \frac{\sqrt{n}}{2}\right) &= P\left(\bigcup_{i=0}^{\sqrt{n}/2-1} \mathcal{E}_i\right) \\ &\le \sum_{i=0}^{\sqrt{n}/2-1} P(\mathcal{E}_i) \\ &\le \sum_{i=0}^{\sqrt{n}/2-1} \frac{i}{n} \\ &= \frac{1}{n} \cdot \binom{\sqrt{n}/2}{2} < \frac{1}{6}. \end{aligned}$$

After $\sqrt{n}$ samples, each additional sample is a repetition with probability at least $1/\sqrt{n}$. The number of iterations before the algorithm returns is thus stochastically dominated by $\sqrt{n}+Geom(1/\sqrt{n})$. We may thus bound the expectation as

$$E\left[\hat{N}\right] = E\left[4\hat{s}^2\right] \le O\left(n + E\left[Geom(1/\sqrt{n})^2\right]\right) \le O(n)$$

where the last inequality is a standard result on the second moment of the geometric random variable. □

Simulating uniform sampling.

We define buckets $B_i = \{a \in U \mid w(a) \in [2^i, 2^{i+1})\}$ for each $i \in \mathbb{Z}$, and we show how to sample elements uniformly from B_i, while allowed to sample elements proportionally to their weight w. First, we show how to sample elements from a bucket B_b given a $b \in \mathbb{Z}$ proportionally in PropBucketSampler. We then use rejection sampling to obtain a uniform sample through UnifBucketSampler.

Lemma 6. *Let $p_b = P(w(a) \in [2^b, 2^{b+1}))$ for $a \in U$ sampled proportionally. Then, the expected sample complexity of both* PropBucketSampler *and* UnifBucketSampler *is $O(1/p_b)$.* PropBucketSampler *returns an item from the b-th bucket with distribution proportional to the weights.* UnifBucketSampler *returns an item from the b-th bucket distributed uniformly.*

Algorithm 3: PropBucketSampler(b)

```
1 Sample $(a, w(a))$ proportionally
2 while $w(a) \notin [2^b, 2^{b+1})$ do
3     Sample $(a, w(a))$ proportionally
4 return $(a, w(a))$
```

Algorithm 4: UnifBucketSampler(b)

```
1 $(a, w(a)) \leftarrow$ PropBucketSampler($b$)
2 while Uniform($[0, 1]$) $> \frac{2^b}{w(a)}$ do
3     $(a, w(a)) \leftarrow$ PropBucketSampler($b$)
4 return $(a, w(a))$
```

Proof. PropBucketSampler performs samples until it samples an item a from bucket B_b; it returns a. This is equivalent to sampling proportionally conditioned on $a \in B_b$. This proves that the output has the claimed distribution. In each step, we finish with probability p_b, independent of other steps. The expected number of steps is, therefore, $1/p_b$. This proves the sample complexity.

Similarly, we terminate UnifBucketSampler after sampling $a \in B_b$ and Uniform($[0, 1]$) $\leq \frac{2^b}{w(a)}$. Sampling until $a \in B_b$ is equivalent to sampling item a from B_b with probability $w(a)/A_b$ where A_b is the total weight of items in bucket B_b. Therefore, a is sampled in each step with probability

$$p_b \cdot \frac{w(a)}{A_b} \cdot \frac{2^b}{w(a)} = \frac{p_b 2^b}{A_b}$$

Since this probability is the same for all items $a \in B_b$, the resulting distribution is uniform. The rejection probability is upper-bounded by $1/2$. Therefore, UnifBucketSampler also has expected sample complexity of $O(1/p_b)$ □

Putting it together: Estimating W without advice.

Finally, we are ready to show the algorithm NoAdvicePropEstimator, that estimates W without relying on any advice $\tilde{n} \geq n$. To analyze it, we first need a lemma.

Algorithm 5: NoAdvicePropEstimator(ε)

```
1 Sample $(a_1, w(a_1)), (a_2, w(a_2))$ proportionally
2 $b_i \leftarrow \lfloor \log w(a_i) \rfloor$ for $i = 1, 2$
3 $b \leftarrow \max(b_1, b_2)$
4 $\tilde{n}_b \leftarrow$ SetSizeEstimator() using UnifBucketSampler($b$) as sampling subroutine, with
   success probability 9/10
5 $\hat{W}_b \leftarrow$ PropEstimator($\frac{\varepsilon}{3}, \tilde{n}_b$) using PropBucketSampler($b$) as sampling subroutine,
   with success probability 9/10
6 $\hat{P}_b \leftarrow$ BernoulliEstimator $\left(\frac{\varepsilon}{3}\right)$ to estimate $P(a \in B_b)$ with success probability 9/10
7 $\hat{W} \leftarrow \hat{W}_b / \hat{P}_b$
8 return $\hat{W}$
```

To analyze NoAdvicePropEstimator, we first need a lemma:

Lemma 7. *Consider b_1, b_2 and b as defined in* NoAdvicePropEstimator, *and let $a \in U$ be a random element sampled proportionally. Then,*

$$E\left[\frac{1}{P_{prop}\left(a \in B_b \mid b\right)}\right] = O(\log n).$$

Moreover, if we define n_b as the number of items in B_b we have

$$E\left[\frac{\sqrt{n_b}}{P_{prop}\left(a \in B_b \mid b\right)}\right] = O(\sqrt{n}).$$

.

Proof. Throughout this proof, we assume a to be sampled proportionally. We first prove the first statement. Define $k = \max\{j \mid B_j \neq \emptyset\}$, and set $\mathcal{B} = \{B_j | k - 2\log n < j \leq k \text{ and } B_j \neq \emptyset\}$. Notice that $|\mathcal{B}| \leq 2\log n$ and, for each $B_j \notin \mathcal{B}$, $x \in B_j$ we have

$$P\left(a = x\right) \leq \frac{2^{k-2\log n+1}}{W} = 2^{-2\log n+1} \cdot \frac{2^k}{W} \leq \frac{2}{n^2}$$

since there exists $y \in B_k$ and therefore $W \geq w(y) \geq 2^k$. Hence, sampling $a \in U$ proportionally we have

$$\begin{aligned} P\left(B_a \notin \mathcal{B}\right) &= P\left(\exists j, \text{ s.t. } a \in B_j \wedge B_j \notin \mathcal{B}\right) \\ &\leq \sum_{x \in \bigcup_{B_j \notin \mathcal{B}} B_j} P(a = x) \\ &\leq n \cdot \frac{2}{n^2} = \frac{2}{n}. \end{aligned}$$

Now, we notice that for each B_j, it holds

$$\begin{aligned} P\left(b \in B_j\right) &\leq 2 \cdot P\left(b_1 \in B_j\right) \cdot P\left(\exists i \leq j \,:\, b_2 \in B_i\right) \qquad (1) \\ &= 2 \cdot P\left(a \in B_j\right) \cdot P\left(\exists i \leq j \,:\, a \in B_i\right) \\ &\leq 2 \cdot P\left(a \in B_j\right) \end{aligned}$$

where the factor two is given by the union bound, and we used $a \sim b_1 \sim b_2$. Then, we can write

$$\begin{aligned} E\left[\frac{1}{P\left(a \in B_b \mid b\right)}\right] &= \sum_{B_j \neq \emptyset} \frac{P\left(b \in B_j\right)}{P\left(a \in B_j\right)} \\ &= \sum_{B_j \in \mathcal{B}} \frac{P\left(b \in B_j\right)}{P\left(a \in B_j\right)} + \sum_{\substack{B_j \notin \mathcal{B} \\ B_j \neq \emptyset}} \frac{P\left(b \in B_j\right)}{P\left(a \in B_j\right)} \\ &\leq 2 \cdot |\mathcal{B}| + \sum_{\substack{B_j \notin \mathcal{B} \\ B_j \neq \emptyset}} 2P\left(\exists i \leq j \,:\, a \in B_i\right) \\ &\leq 2 \cdot |\mathcal{B}| + \sum_{\substack{B_j \notin \mathcal{B} \\ B_j \neq \emptyset}} 2P\left(B_a \notin \mathcal{B}\right) \\ &\leq 4\log n + 2n \cdot P\left(B_a \notin \mathcal{B}\right) \\ &\leq 4\log n + 4 = O(\log n). \end{aligned}$$

The fist inequality is obtained using eq. (1). The second inequality descends from the fact that $B_j \notin \mathcal{B}$ and $i \leq j$ imply $B_i \notin \mathcal{B}$. The last two inequalities are obtained using n as an upper bound on the number of nonempty buckets B_j and recalling that $P(B_a \notin \mathcal{B}) \leq 2/n$.

Now we can prove the second statement. Denote by n_b the number of elements in B_b and define $\ell = \arg\max_{i \in \mathbb{Z}} n_i 2^{i/2}$. If we define $S_i = \sum_{j \leq i} \sum_{a \in B_j} w(a)$, then we can rewrite the already proven inequality $P\left(b \in B_j\right) \leq 2 \cdot P\left(a \in B_j\right) \cdot P\left(\exists i \leq j \,:\, a \in B_i\right)$ as

$$P\left(b \in B_j\right) \leq 2 \cdot P\left(a \in B_j\right) \cdot \frac{S_j}{W}. \tag{2}$$

We now prove that there exists a constant $C > 0$ such that, for all $i \in \mathbb{Z}$ it holds $S_{\ell-i} \leq C \cdot S_\ell \cdot 2^{-i/2}$. Notice that, by definition of ℓ, we have $n_j \cdot 2^{j/2} \leq n_\ell \cdot 2^{\ell/2}$ for all $j \in \mathbb{Z}$. We can now bound

$$\begin{aligned} S_{\ell-i} &\leq \sum_{j \leq \ell-i} n_j \cdot 2^{j+1} \\ &\leq n_\ell \cdot \sum_{j \leq \ell-i} 2^{\ell/2 - j/2} \cdot 2^{j+1} \\ &= 2 n_\ell 2^\ell \cdot \sum_{j \leq \ell-i} 2^{j/2-\ell/2} \\ &\leq 2 S_\ell \cdot \sum_{j \leq \ell-i} 2^{j/2-\ell/2} \leq C \cdot S_\ell 2^{-i/2}. \end{aligned}$$

Therefore, we have $\sum_{j<\ell} S_j = O(S_\ell)$. Notice that by the definition of ℓ we have $n_{\ell+i} \leq n_\ell \cdot 2^{-i/2}$ for each $i \geq 0$. Now we are ready to prove our final result.

$$\begin{aligned} E\left[\frac{\sqrt{n_b}}{P\left(a \in B_b \,|\, b\right)}\right] &= \sum_{j \in \mathbb{Z}} P\left(b \in B_j\right) \cdot \frac{\sqrt{n_j}}{P\left(a \in B_j\right)} \\ &\leq \sum_{j \in \mathbb{Z}} 2 \frac{S_j}{W} \cdot \sqrt{n_j} \\ &\leq \sum_{j < \ell} 2 \frac{S_j}{W} \cdot \sqrt{n} + \sum_{j \geq \ell} 2 \sqrt{n_j} \\ &\leq \frac{O(S_\ell)}{W} \cdot \sqrt{n} + \sum_{i \geq 0} 2 \sqrt{n_\ell} \cdot 2^{-i/4} = O(\sqrt{n}). \end{aligned}$$

The first inequality uses eq. (2), the second inequality is obtained splitting the series in two parts and using $S_j \leq W$. The last inequality is obtained plugging in $\sum_{j<\ell} S_j = O(S_\ell)$ and $n_{\ell+i} \leq n_\ell \cdot 2^{-i/2}$ for $i \geq 0$. □

Now we are ready to analyze NoAdvicePropEstimator.

Theorem 8. *Let $\hat{W}$ be the estimate returned by* NoAdvicePropEstimator. *Then $\hat{W}$ is an $(1 \pm \varepsilon)$-approximation of W with probability $2/3$. Moreover, its expected sample complexity is*

$$O\left(\frac{\sqrt{n}}{\varepsilon} + \frac{\log(n)}{\varepsilon^2}\right).$$

Proof. We start by proving correctness. Define $W_b = \sum_{x \in B_b} w(x)$ and $P_b = P(a \in B_b \,|\, b) = W_b / W$. Notice that W_b and P_b are random variables, since they depend on b. Now we prove that $\hat{W}$ is a $(1 \pm \varepsilon)$-approximation of W with probability 2/3. Define the event $\mathcal{E}_1 = \{n \leq \tilde{n}_b\}$, we have $P(\mathcal{E}_1) \geq 9/10$ (where we use Theorem 5 together with probability amplification to amplify the success probability of 2/3 to 9/10). Define the event

$$\mathcal{E}_2 = \Big\{(1-\varepsilon/3)W_b \leq \hat{W}_b \leq (1+\varepsilon/3)W_b\Big\}$$

then, we have $P(\mathcal{E}_2 \,|\, \mathcal{E}_1) \geq 9/10$ (where we use Theorem 4 and probability amplification). Define the event

$$\mathcal{E}_3 = \Big\{(1-\varepsilon/3)P_b \leq \hat{P}_b \leq (1+\varepsilon/3)P_b\Big\}$$

then it holds $P(\mathcal{E}_3 \,|\, b) \geq 9/10$ (where we use Theorem 1 and probability amplification). On the event $\mathcal{E}_2 \cap \mathcal{E}_3$, it holds

$$(1-\varepsilon)\cdot W \leq \frac{1-\varepsilon/3}{1+\varepsilon/3}\cdot\frac{W_b}{P_b} \leq \frac{\hat{W}_b}{\hat{P}_b} \leq \frac{1+\varepsilon/3}{1-\varepsilon/3}\cdot\frac{W_b}{P_b} \leq (1+\varepsilon)\cdot W.$$

Then we can apply union bound and prove

$$\begin{aligned} P\Big(\hat{W} < (1-\varepsilon)W \text{ or } \hat{W} > (1+\varepsilon)W\Big) &\leq \\ P\left(\bar{\mathcal{E}}_2 \cup \bar{\mathcal{E}}_3\right) &\leq \\ P\left(\bar{\mathcal{E}}_1\right) + P\left(\bar{\mathcal{E}}_2 \,|\, \mathcal{E}_1\right) + P(\bar{\mathcal{E}}_3) &\leq \\ \frac{1}{10}+\frac{1}{10}+\frac{1}{10} &\leq \frac{1}{3}. \end{aligned}$$

It remains to prove that the expected number of samples that NoAdvicePropEstimator uses is as claimed. Denote by σ_1 the total number of samples taken on line 4, by σ_2 the total number of samples taken on line 5 and by σ_3 the total number of samples taken on line 6. We denote the number of samples employed during the i-th call to UnifBucketSampler(b) on line 4 with $\eta_1^{(i)}$; similarly, we denote the number of samples taken during the i-th call to PropBucketSampler(b) on line 5 with $\eta_2^{(i)}$. We can then write

$$\sigma_1 = \sum_{i=1}^{\tau_1} \eta_1^{(i)} \text{ and } \sigma_2 = \sum_{i=1}^{\tau_2} \eta_2^{(i)}$$

where τ_1 and τ_2 are the number of calls to UnifBucketSampler(b) performed on line 4 and line 5, respectively. First we notice that, thanks to Lemma 6, there exists a constant $K > 0$ such that $E[\eta_1^{(i)} \,|\, b], E[\eta_2^{(i)} \,|\, b] \leq \frac{K}{P(a\in B_b \,|\, b)}$. Thanks to Theorem 5, we have $E[\tau_1 \,|\, b] = O(\sqrt{n_b})$ and $\tau_2 = O(\sqrt{\tilde{n}_b}/\varepsilon)$. Now we are ready to bound $E[\sigma_1]$ and $E[\sigma_2]$. We have

$$\begin{aligned} E\left[\sigma_1\right] &= E\left[E\left[\sum_{i=1}^{\tau_1}\eta_1^{(i)} \,|\, b\right]\right] \\ &= E\left[E\left[\tau_1 \,|\, b\right]\cdot E\left[\eta_1^{(1)} \,|\, b\right]\right] \\ &\leq E\left[\frac{K\cdot O(\sqrt{n_b})}{P(a \in B_b \,|\, b)}\right] = O\left(\sqrt{n}\right) \end{aligned}$$

where the first equality is by the Wald's identity and the last equality is obtained applying Lemma 7. Similarly,

$$\begin{aligned}
E\left[\sigma_2\right] &= E\left[E\left[\sum_{i=1}^{\tau_2}\eta_2^{(i)} \mid b\right]\right] \\
&= E\left[E\left[\tau_2 \mid b\right] \cdot E\left[\eta_2^{(1)} \mid b\right]\right] \\
&\le E\left[\frac{K \cdot \left(\sqrt{80\tilde{n}_b}/\varepsilon + 1\right)}{P(a \in B_b \mid b)}\right] \\
&= O\left(\frac{\sqrt{n}}{\varepsilon} + \log n\right) = O\left(\frac{\sqrt{n}}{\varepsilon}\right)
\end{aligned}$$

where we used that $E[\sqrt{\tilde{n}_b}] \le \sqrt{E[\tilde{n}_b]} \le \sqrt{n}$, which holds thanks to Theorem 5 and Jensen inequality. In order to bound $E[\sigma_3 \mid b]$, recall that, thanks to Theorem 1, there exists a $C > 0$ such that, conditioning on the value of b, BernoulliEstimator($\frac{\varepsilon}{3}$) takes in expectation at most $\frac{C}{P(a \in B_b \mid b)\varepsilon^2}$ samples in order to estimate $P(a \in B_b)$. Therefore we have

$$E[\sigma_3] = E\left[E[\sigma_3 \mid b]\right] \le E\left[\frac{C}{P\left(a \in B_b \mid b\right)\varepsilon^2}\right] = O\left(\frac{\log n}{\varepsilon^2}\right).$$

where the last equality holds by Theorem 7. This concludes the proof, since the total number of samples taken by NoAdvicePropEstimator is $\sigma_1 + \sigma_2 + \sigma_3$. By the bounds we have proven above, the expectation of $\sigma_1 + \sigma_2 + \sigma_3$ is as claimed. $\square$

3 Sum Estimation by Hybrid Sampling.

In this section, we assume that we can sample elements both proportionally and uniformly. Again, we solve the task of providing an estimate $\hat{W}$ of W such that $\hat{W}$ is a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$.

We notice that if we take $\Theta(n \log n)$ uniform samples then, with probability $2/3$, we see every element of U. This simple analysis is well-known under the name of Coupon Collector problem. If we know n (or any constant-factor approximation of it) we may simply take $\Theta(n \log n)$ samples, assume we have seen every element at least once, and compute W exactly. However, if we do not know n, a more complex scheme is required to achieve a complexity of $O(n \log n)$, which we describe in Section 3.2. Therefore, it is sufficient to show an algorithm with complexity $T(n, \varepsilon)$ to obtain a complexity of the form $O(\min(T(n, \varepsilon), n \log n))$, as we can just run the coupon-collector algorithm in parallel and take the result provided by the first of the two algorithms to finish its execution. In what follows we only show how to achieve a complexity of $O(\sqrt[3]{n}/\varepsilon^{4/3})$ when $|U|$ is known, and of $O(\sqrt{n}/\varepsilon)$ when $|U|$ is unknown. As a consequence, the complexities that we achieve in this settings are $O(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n \log n))$ and of $O(\min(\sqrt{n}/\varepsilon, n \log n))$ respectively.

3.1 Algorithms for $|U|$ known.

In this section, we show an algorithm that, given $n = |U|$, returns a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$ using $O(\sqrt[3]{n}/\varepsilon^{4/3})$ samples. First, we introduce a subroutine that uses harmonic mean to estimate the average weight W/n; then we combine it with PropEstimator to obtain the main algorithm of this section.

Harmonic-mean-based estimator.

Here we show the algorithm HarmonicEstimator($\varepsilon, \tilde{\theta}, \phi$) that returns an $(1 \pm \varepsilon)$-approximation $\hat{\theta}$ of W/n with probability 2/3. HarmonicEstimator($\varepsilon, \tilde{\theta}, \phi$) takes as advice an upper bound on the average weight $\tilde{\theta} \geq W/n$, and a parameters ϕ such that we expect $P_{unif}(w(a) \geq \phi)$ not to be too small and $\tilde{\theta}/\phi$ not to be too large. A more formal statement follows.

```
Algorithm 6: HarmonicEstimator($\varepsilon, \tilde{\theta}, \phi$)
1  $\hat{p} \leftarrow$ BernoulliEstimator $\left(P_{unif}(w(a) \geq \phi), \frac{\varepsilon}{3}\right)$ with success probability 9/10
2  $k \leftarrow 45 \cdot \frac{\tilde{\theta}}{\phi(1-\varepsilon/3)\hat{p}\varepsilon^2}$
3  Sample $a_1 \dots a_k$ proportionally
4  for $i = 1 \dots k$ do
5      if $w(a_i) \geq \phi$ then
6          $b_i = 1/w(a_i)$
7      else
8          $b_i = 0$
9  $H = \sum_{i=1}^{k} b_i/k$
10 $\hat{\theta} \leftarrow \hat{p}/H$
11 return $\hat{\theta}$
```

To see the intuition behind this algorithm, consider the case when $\phi \leq w(a)$ for all $a \in U$. It then holds $\hat{p} \approx 1$. We take k samples, and let $1/H$ be the harmonic mean of the weights of the sampled items. We have $E[H] = n/W$, and $\hat{\theta} \approx 1/H$ as $\hat{p} \approx 1$. Unfortunately H might have a high variance due to elements having very small weights. To fix this, we consider a parameter ϕ such that $w(a) < \phi$ for some $a \in U$. Instead of $E[H] = n/W$, we then have $E[H] = n'/W$ for $n' = |\{a \in U \mid w(a) \geq \phi\}|$. We then multiply $1/H$ by $\hat{p}$ in order to adjust for the fraction of items that were ignored. Note that, while increasing ϕ, the variance of $1/H$ decreases; however, also n' decreases and this means that computing an estimate $\hat{p} \approx n'/n$ requires more samples. This introduces a trade-off between the algorithm's complexity and the variance of H.

Lemma 9. *Given parameters $\tilde{\theta}$ and $0 < \varepsilon < 1$, HarmonicEstimator($\varepsilon, \tilde{\theta}, \phi$) has expected sample complexity $O((1 + \frac{\tilde{\theta}}{\phi})/(p\,\varepsilon^2))$ where $p = P_{unif}(w(a) \geq \phi)$. It returns an estimate $\hat{W}$ such that $P(\hat{\theta} < W/(20n)) \leq 1/20$. If, moreover, $\tilde{\theta} \geq W/n$, then $P(|\hat{W} - W| \leq \varepsilon W) \geq 2/3$.*

Proof. We start by proving that $\hat{\theta}$ is a $(1 + \varepsilon)$-approximation of W/n with probability 2/3 when $\tilde{\theta} \geq W/n$. Define the event $\mathcal{E} = \{\hat{p}$ is a $(1 \pm \varepsilon/3)$-approximation of $p\}$. By Theorem 1 and using probability amplification, we have $P(\mathcal{E}) \geq 9/10$. For each $i = 1 \dots k$ we have

$$E[b_i] = \sum_{\substack{a \in U, \\ w(a) \geq \phi}} \frac{1}{w(a)} \cdot \frac{w(a)}{W} = \frac{n_{\geq \phi}}{W} = \frac{p \cdot n}{W}$$

where $n_{\geq \phi}$ is the number of elements in $a \in U$ with $w(a) \geq \phi$. Notice that this implies $E[H] =$

$p \cdot n/W$. Moreover, for each $i = 1 \dots k$

$$Var(b_i) \leq E\left[b_i^2\right] = \\ \sum_{\substack{a \in U, \\ w(a) \geq \phi}} \frac{1}{w^2(a)} \cdot \frac{w(a)}{W} \leq \\ \frac{n_{\geq \phi}}{\phi \cdot W} = \frac{p \cdot n}{\phi \cdot W}.$$

Conditioning on $\mathcal{E}$, we have that $(1 - \varepsilon/3)\hat{p} \leq p$. The way we have set k allows us to bound

$$Var(H \,|\, \mathcal{E}) = \frac{Var(b_i)}{k} = \\ \frac{p \cdot n}{\phi \cdot W} \cdot \frac{\varepsilon^2 (1 - \varepsilon/3)\hat{p}}{45} \cdot \frac{\phi}{\tilde{\theta}} \leq \\ \left(\frac{p \cdot n}{W}\right)^2 \cdot \frac{\varepsilon^2}{45} = E[H]^2 \cdot \frac{\varepsilon^2}{45}$$

where we used that $\tilde{\theta} \geq W/n$. It holds $E[H \,|\, \mathcal{E}] = E[H]$. We may thus apply Chebyshev's inequality to get

$$P\left(|H - E[H]| > \frac{\varepsilon}{3} E[H] \,\middle|\, \mathcal{E}\right) \leq \frac{Var(H \,|\, \mathcal{E})}{\left(\frac{\varepsilon}{3} E[H]\right)^2} \leq \frac{9}{45} = \frac{1}{5}$$

Since $\varepsilon < 1$, we have $(1 - \varepsilon/3)^{-1} \leq 1 + \varepsilon/2$ and $(1 + \varepsilon/3)^{-1} \geq 1 - \varepsilon/2$. Therefore

$$P\left(\left|\frac{1}{H} - \frac{1}{E[H]}\right| > \frac{\varepsilon}{2} \cdot \frac{1}{E[H]} \,\middle|\, \mathcal{E}\right) \leq \frac{1}{5}.$$

Again, since $\varepsilon < 1$, we have $(1 + \varepsilon/3) \cdot (1 + \varepsilon/2) \leq 1 + \varepsilon$ and $(1 - \varepsilon/3) \cdot (1 - \varepsilon/2) \geq 1 - \varepsilon$. Hence, using the union bound

$$P\left(\left|\frac{\hat{p}}{H} - \frac{p}{E[H]}\right| > \varepsilon \cdot \frac{p}{E[H]}\right) \leq \\ P(\bar{\mathcal{E}}) + P\left(\left|\frac{1}{H} - \frac{1}{E[H]}\right| > \frac{\varepsilon}{2} \cdot \frac{1}{E[H]} \,\middle|\, \mathcal{E}\right) \leq \frac{1}{3}.$$

Since $p/E[H] = W/n$, we have that the estimate $\hat{\theta} = \hat{p}/H$ is a $(1 + \varepsilon)$-approximation of W/n with probability $\geq 2/3$.

We now argue the sample complexity. The expected number of samples used on line 1 is by Theorem 1 equal to $O(1/(p\,\varepsilon^2))$. In the rest of the algorithm, we use k samples. It holds

$$E[k] = E\left[\frac{1}{\hat{p}}\right] \cdot O\left(\frac{\tilde{\theta}}{\phi\,\varepsilon^2}\right) = O\left(\frac{\tilde{\theta}}{p\,\phi\,\varepsilon^2}\right)$$

where the second equality holds by Theorem 1. The sample complexity is thus as claimed.

Finally, we prove that, regardless of $\tilde{\theta}$, it holds $P(\hat{\theta} < W/(20n)) \leq 1/20$. It holds $E[H/\hat{p}] = E[H]E[1/\hat{p}] = n/W$ where $E[1/\hat{p}] = 1/p$ by Theorem 1. Therefore, by the Markov's inequality

$$P\left(\frac{\hat{p}}{H} \leq \frac{W}{20n}\right) = P\left(\frac{H}{\hat{p}} \geq \frac{20n}{w}\right) \leq \frac{1}{20}.$$

□

Combining the two algorithms.

Here, we combine HarmonicEstimator with PropEstimator to obtain HybridEstimator. It works in the hybrid setting and provides a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$, using in expectation $O(\sqrt[3]{n}/\varepsilon^{4/3})$ samples. While analysing HybridEstimator, we can restrict ourselves to $\varepsilon \geq 8/\sqrt{n}$. Indeed, for very small values of ε (namely, $\varepsilon \leq 1/(\sqrt{n}\log n)$) we use the coupon-collector algorithm, and for intermediate values of ε (namely, $1/(\sqrt{n}\log n) < \varepsilon < 8/\sqrt{n}$) we use PropEstimator$(n, \varepsilon)$. The coupon collector algorithm gives a sample complexity of $O(n \log n)$, that is better than $O(\sqrt[3]{n}/\varepsilon^{4/3})$ for $\varepsilon < 1/(\sqrt{n}\log n)$. PropEstimator gives a sample complexity of $O(\sqrt{n}/\varepsilon)$, that is better than $O(\sqrt[3]{n}/\varepsilon^{4/3})$ for $\varepsilon < 8/\sqrt{n}$.

Algorithm 7: HybridEstimator(n, ε)

```
Abort this algorithm if it uses more than $C \frac{n^{1/3}}{\varepsilon^{4/3}}$ samples, where $C$ is a large enough
 constant.
1  Find $\theta$ such that $P\left(\frac{n^{2/3}}{\varepsilon^{2/3}} \leq \left|\{a \in U \mid w(a) \geq \theta\}\right| \leq 2 \cdot \frac{n^{2/3}}{\varepsilon^{2/3}}\right) \geq \frac{19}{20}$
2  $\hat{p} \leftarrow$ BernoulliEstimator $\left(P_{prop}(w(a) \geq \theta), \frac{\varepsilon}{3}\right)$ with success probability 19/20
3  if $\hat{p} \geq 1/2$ then
4      $\tilde{n}_{\geq\theta} \leftarrow 2 \cdot \frac{n^{2/3}}{\varepsilon^{2/3}}$
5      $\hat{W}_{\geq\theta} \leftarrow$ PropEstimator $\left(\tilde{n}_{\geq\theta}, \frac{\varepsilon}{3}\right)$ with success probability 19/20, run on proportional
        samples conditioned on $w(a) \geq \theta$, obtained by rejecting elements with $w(a) < \theta$
6      $\hat{W} \leftarrow \hat{W}_{\geq\theta}/\hat{p}$
7  else
8      $\hat{p} \leftarrow$ HarmonicEstimator$(\varepsilon, 3\theta, \theta)$ with success probability 19/20
9      $\hat{W} \leftarrow n \cdot \hat{p}$
10 return $\hat{W}$
```

To implement line 1 it is sufficient to sample uniformly $120\, n^{1/3}\varepsilon^{2/3}$ elements of U and define θ as the element with the 180-th largest weight. (Note that for $\varepsilon \geq 8/\sqrt{n}$ we have $120\, n^{1/3}\varepsilon^{2/3} \geq 480$, so this is well-defined.) Let k be such that there are k elements $a \in U$ with $w(a) \geq \theta$. A standard analysis using Chebyshev inequality shows that k is concentrated around $\frac{3}{2}\frac{n^{2/3}}{\varepsilon^{2/3}}$.

Theorem 10. *Given ε such that $8/\sqrt{n} \leq \varepsilon < 1$,* HybridEstimator$(n, \varepsilon)$ *uses $O(\sqrt[3]{n}/\varepsilon^{4/3})$ samples. With probability at least $2/3$, the returned estimate $\hat{W}$ is a $(1 \pm \varepsilon)$-approximation of W.*

Proof. We first analyze a variant of the algorithm that does not abort. (That is, an algorithm with the same pseudocode except for the abortion condition removed.) We now show that this algorithm returns a $(1 \pm \varepsilon)$-approximation with probability at least $5/6$.

The scheme summarized above to find θ at line 1 succeeds with probability at least $19/20$. We call this event $\mathcal{E}_1$. Define the event $\mathcal{E}_2 = \{\hat{p}$ is a $(1 \pm \varepsilon/3)$-approximation of $p\}$. It holds $P(\mathcal{E}_2) \geq 19/20$.

We now consider the case $\hat{p} \geq 1/2$. On line 5 we employ the algorithm PropEstimator. Whenever it performs a sample, we simulate a proportional sample from the set $U_\theta = \{a \in U \mid w(a) \geq \theta\}$ by sampling until we sample item a such that $w(a) \geq \theta$. It is easy to see that the distribution obtained with this sampling scheme is exactly the proportional distribution on the set U_θ. Conditioning on $\mathcal{E}_1$, $\tilde{n}_{\geq\theta}$ is a valid advice and (by Theorem 4) PropEstimator returns a $(1 \pm \varepsilon/3)$-approximation of $W_\theta = \sum_{w(a)\geq\theta} w(a)$ with probability at least $2/3$. We amplify this probability to

19/20. Hence, we have

$$P\left(\left\{\hat{W}_\theta \text{ is a } (1\pm\varepsilon)\text{-approximation of } W_{\geq\theta}\right\} \cap \mathcal{E}_2\right) \geq \\ 1 - \left(\frac{1}{20} + P(\bar{\mathcal{E}_1}) + P(\bar{\mathcal{E}_2})\right) \geq \frac{5}{6}.$$

On this event, since $\varepsilon < 1$, we have

$$(1-\varepsilon)W \leq \frac{1-\varepsilon/3}{1+\varepsilon/3}W \leq \hat{W} \leq \frac{1+\varepsilon/3}{1-\varepsilon/3}W \leq (1+\varepsilon)W.$$

Now we analyse the case $\hat{p} < 1/2$. Whenever $3\theta \geq W/n$, HarmonicEstimator($\varepsilon, 3\theta, \theta$) returns a $(1\pm\varepsilon)$-approximation of W/n with probability 2/3 (by Theorem 9). We amplify that probability to 19/20. Now we argue that, conditioning on $\mathcal{E}_2$, we have $3\theta \geq W/n$. Define $p = P_{prop}(w(a) \geq \theta)$, then (on $\mathcal{E}_2$) we have $p \leq (1+\epsilon/3)\hat{p} \leq (1+1/3)1/2 \leq 2/3$. Hence, $n\theta \geq (1-p)W \geq W/3$ and thus $3\theta \geq W/n$, where the first inequality is obtained using $\sum_{w(a)<\theta} w(a) = (1-p)W$. Applying union bound gives that HarmonicEstimator($\varepsilon, 3\theta, \theta$) succeeds with probability at least $1 - (1/20 + P(\bar{\mathcal{E}_2})) \geq 5/6$.

Therefore, regardless of the value of $\hat{p}$, we have shown that $\hat{W}$ is a $(1\pm\varepsilon)$-approximation of W with probability at least 5/6. Thus, the modified algorithm without abortion is correct with probability at least 5/6. We now argue that the probability that Algorithm 7 is aborted is at most 1/6 (for C large enough). By the union bound, its success probability is at least 2/3.

In what follows, we compute how many samples are taken on each line. On line 1, we use only $120n^{1/3}\varepsilon^{2/3}$ samples. Thanks to Theorem 1, BernoulliEstimator on line 2 uses $O(1/(p\varepsilon^2))$ samples in expectation. In what follows, we condition on $\mathcal{E}_1 \cap \mathcal{E}_2$. It holds $p \geq P_{unif}(w(a) \geq \theta)$ and thus we have $p \geq \frac{1}{\varepsilon^{2/3}n^{1/3}}$. The number of samples used on line 2 is then in expectation $O(n^{1/3}/\varepsilon^{4/3})$. By the (conditional) Markov's inequality the probability that we use more than $C_1 n^{1/3}/\varepsilon^{4/3}$ is at most 1/30, for some C_1 large enough. PropEstimator uses $O(\sqrt{n^{2/3}/\epsilon^{2/3}}/\epsilon) = O(n^{1/3}/\varepsilon^{4/3})$ samples in the worst case. The rejections cause only constant factor expected slowdown. Again, by the (conditional) Markov's inequality, for C_2 large enough, we use more than $C_2 n^{1/3}/\varepsilon^{4/3}$ with probability at most 1/30. Since we have $p \geq \frac{1}{\varepsilon^{2/3}n^{1/3}}$, on line 8 HarmonicEstimator takes $O(n^{1/3}/\varepsilon^{4/3})$ samples in expectation (by Theorem 9). Thus, there exists a constant C_3 such that on line 8 we use more than $C_3 n^{1/3}/\varepsilon^{4/3}$ samples with probability at most 1/30.

Set $C = 120 + C_1 + C_2 + C_3$. It then holds by the union bound that we use more than $C\sqrt[3]{n}/\varepsilon^{4/3}$ samples (and thus abort) with probability at most $P(\bar{\mathcal{E}_1}) + P(\bar{\mathcal{E}_2}) + 1/30 + 1/30 = 1/6$. □

3.2 Algorithms for $|U|$ unknown.

In this section we show an algorithm that, without any knowledge of $n = |U|$, provides a $(1\pm\varepsilon)$-approximation of W with probability 2/3 using $O(\sqrt{n}/\varepsilon)$ samples. This complexity is strictly higher than the one of Section 3.1 for $\varepsilon = \omega(1/\sqrt{n})$. However, it is near-optimal when we do not know n, as we will see in Section 4.

Algorithm 8: NoAdviceHybridEstimator(ε)

```
1 $\tilde{n} \leftarrow$ SetSizeEstimator(), with success probability 5/6 (using uniform sampling)
2 $\hat{W} \leftarrow$ PropEstimator($\tilde{n}, \varepsilon$), with success probability 5/6 (using proportional sampling)
3 return $\hat{W}$
```

Theorem 11. NoAdviceHybridEstimator(ε) *uses in expectation* $O(\sqrt{n}/\varepsilon)$ *samples and, with probability at least* 2/3*, we have* $(1-\varepsilon)W \le \hat{W} \le (1+\varepsilon)W$.

Proof. By Theorem 5, SetSizeEstimator takes $O(\sqrt{n})$ samples and returns $\tilde{n}$ such that $n \le \tilde{n}$ with probability 2/3. We amplify this probability to 5/6. Conditioning on $n \le \tilde{n}$, PropEstimator returns $(1 \pm \varepsilon)$-estimate of W with probability at least 2/3 by Theorem 4. We amplify this probability to 5/6. By the union bound, the algorithm returns a $(1 \pm \varepsilon)$-estimate with probability at least 2/3.

Moreover, by Theorem 5 and Theorem 2, $E[\tilde{n}] = O(n)$. By the Jensen inequality, PropEstimator then uses in expectation $O(\sqrt{n}/\varepsilon)$ samples. □

Coupon collector algorithm.

In this section, we give an algorithm that returns W *exactly* in time $O(n \log n)$, with probability at least 2/3. In fact, we show that we may learn the whole set U, along with all the weights, with probability 2/3 in this sample complexity. From this, the sum can be easily computed. We can use this to ensure we never use more than $O(n \log n)$ samples in the hybrid setting. Specifically, we may execute in parallel this algorithm together with one of the above algorithms and abort when one of them returns an estimate. If one wishes to implement this in practice, it is possible to instead do the following. Given the parameter ε, compute a threshold n_0 such that we would like to (if we knew n) execute NoAdviceCouponCollector if $n \le n_0$ and NoAdviceHybridEstimator if $n > n_0$. We then run NoAdviceCouponCollector and we abort if in case we find n_0 distinct elements. If this happens, we then run NoAdviceHybridEstimator.

Algorithm 9: NoAdviceCouponCollector()

```
1  $S \leftarrow \emptyset$
2  $k = 0$
3  while $k < 4|S| \log 3|S|$ do
4      Sample $a \in U$ uniformly
5      if $k \notin S$ then
6          $k \leftarrow 0$
7          $S \leftarrow S \cup \{a\}$
8      else
9          $k \leftarrow k + 1$
10 return $\sum_{a \in S} w(a)$
```

Theorem 12. NoAdviceCouponCollector *has expected sample complexity* $O(n \log n)$ *and returns, with probability at least* 2/3 *the whole set* U.

Proof. Thanks to the analysis of the standard coupon collector problem, we know that it takes in expectation $O(n \log n)$ samples before $S = U$. After that, we spend no more than $4|S| \log 3|S| = n \log n$ additional samples. The sample complexity is thus as claimed. It remains to argue correctness.

The returned result is not correct if the algorithm returns S too early. This happens exactly if there exists some ℓ such that it takes more than $4\ell \log 3\ell$ samples to get the $(\ell+1)$-th element. At step ℓ, there are $n - \ell$ elements that are not in S, hence the probability that none of the $4\ell \log 3\ell$

elements fall in the set $U \setminus S$ is

$$(\ell/n)^{4\ell\log 3\ell} \le (1-\frac{1}{2\ell})^{4\ell\log 3\ell} \le (1/e)^{2\log 3\ell} = \frac{1}{9\ell^2}.$$

The first inequality holds since for $\ell \in [1, n-1]$, it holds $\ell/n \le 1 - 1/2\ell$. Taking the union bound over all $1 \le \ell \le n-1$, we have that the failure probability is upper-bounded by $\sum_{\ell=1}^{\infty} \frac{1}{9\ell^2} < 1/3$. $\square$

4 Lower Bounds.

In this section, we give lower bounds for the problems we study in this paper. The proofs of the lower bounds all follow a common thread. In what follows, we use the term *risk* to refer to the misclassification probability of a classifier.

The roadmap of all our proofs follows. First, we define two different instances of the problem (U_1, w_1) and (U_2, w_2) such that a $(1 \pm \varepsilon)$-approximation of W is sufficient to distinguish between them. Then, we define our *hard* instance as an equally likely mixture of the two, namely we take (U_1, w_1) with probability 1/2 and (U_2, w_2) otherwise. We denote these events by $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively. Second, we show that a Bayes classifier[8] cannot distinguish between the two cases with probability 2/3 using too few samples; since Bayes classifiers are risk-optimal this implies that no classifier can have risk less than 2/3 while using the same number of samples. To show that a Bayes classifier has a certain risk, we study the posterior distribution. Let S represent the outcome of the samples. Since the prior is uniform, applying Byes theorem gives

$$\frac{P(\mathcal{E}_1 \mid S)}{P(\mathcal{E}_2 \mid S)} = \frac{P(S \mid \mathcal{E}_1)}{P(S \mid \mathcal{E}_2)}.$$

We call this ratio $\mathcal{R}(S)$ and show that $\mathcal{R}(S) \approx 1$ with probability close to 1. When $\mathcal{R}(X) \approx 1$, the posterior distribution is very close to uniform, and this entails a risk close to 1/2. First, we show this formally with some technical lemmas, and then we instantiate our argument for each of the studied problems.

Lemma 13. *Given two disjoint events $\mathcal{E}_1, \mathcal{E}_2$ such that $P(\mathcal{E}_1) = P(\mathcal{E}_2) = 1/2$ and a random variable X, we define the ratio*

$$\mathcal{R}(X) = \frac{P(X \mid \mathcal{E}_1)}{P(X \mid \mathcal{E}_2)}.$$

Notice that $\mathcal{R}(X)$ is a random variable since it depends on the outcome of X. If

$$P\left(\frac{7}{8} \le \mathcal{R}(X) \le \frac{8}{7}\right) \ge \frac{14}{15}$$

then any classifier taking X and classifying $\mathcal{E}_1, \mathcal{E}_2$ has risk $\ge 2/5$.

Proof. First, we notice that since Bayes classifiers are risk-optimal, then it is sufficient to prove our statement for a Bayes classifier. Define $p_i = P(\mathcal{E}_i \mid X)$ for $i = 1, 2$, then Bayes classifier simply returns $\arg\max_{i=1,2} p_i$. By Bayes theorem and because $P(\mathcal{E}_1) = P(\mathcal{E}_2)$ we have $\mathcal{R}(X) = p_1/p_2$. If $7/8 \le \mathcal{R}(X) \le 8/7$, then

$$\frac{1}{p_2} = \frac{p_1 + p_2}{p_2} = 1 + \mathcal{R}(X) \in \left[\frac{15}{8}, \frac{15}{7}\right]$$

[8]Suppose we have a partition of the probability space Ω into events $\mathcal{E}_1, \cdots, \mathcal{E}_k$. We want to guess which event happened, based on observation X. Bayes classifier outputs as its guess $\mathcal{E}_\ell$ that maximizes $P(\mathcal{E}_\ell | X)$.

and the same holds for p_1, therefore $7/15 \leq p_1, p_2 \leq 8/15$. Hence, conditioning on $7/8 \leq \mathcal{R}(X) \leq 8/7$, the probability of correct classification is at most $8/15$. Finally, we have

$$P\,(\text{Classifier returns correct answer}) \leq$$
$$P\left(\mathcal{R}(X) < \frac{7}{8} \vee \mathcal{R}(X) > \frac{8}{7}\right) + P\left(\text{Classifier returns correct answer} \,\middle|\, \frac{7}{8} \leq \mathcal{R}(X) \leq \frac{8}{7}\right) \leq$$
$$\frac{1}{15} + \frac{8}{15} = \frac{3}{5}.$$

□

Before proving the main theorems, we need several lemmas.

Lemma 14. *Consider an instance of our problem (U, w) so that $n = |U|$ and $w(a) = 1$ for each $a \in U$. We perform m independent samples and denote with ℓ the number of distinct elements obtained. Then, $Var(\ell) \leq m^2/n$. Moreover, for each $\delta > 0$*

$$P\left(|\ell - E[\ell]| \geq \frac{1}{\sqrt{\delta}} \cdot \frac{m}{\sqrt{n}}\right) \leq \delta.$$

Note that, since all weights are the same, the proportional sampling is equivalent to sampling uniformly from U.

Proof. We use the Efron-Stein inequality to prove a bound on the variance of ℓ. Let X_i be the i-th sample. Now ℓ is a function of $X_1, \cdots, X_m$ and we write it as $\ell = f(X_1, \cdots, X_m)$. Let $X'_1, \cdots, X'_m$ be an independent copy of $X_1, \cdots, X_m$. Let $\ell'_i = f(X_1, \cdots, X_{i-1}, X'_i, X_{i+1}, \cdots, X_m)$. It clearly holds that $|\ell - \ell'_i| \in \{0, 1\}$, moreover $\ell - \ell'_i = 1$ if and only if X_i does not collide with with any X_j for $j \neq i$ and X'_i does collide with some X_k for $k \neq i$. It holds that $|\{X_1, \cdots, X_{i-1}, X_{i+1}, \cdots X_m\}| \leq m$. Therefore, the probability that a X'_i lies in this set is $\leq m/n$. Since ℓ and ℓ'_i are symmetric, we have that also $\ell - \ell'_i = -1$ with probability $\leq m/n$. Hence, $E[(\ell - \ell'_i)^2] = P(|\ell - \ell'_i| = 1) \leq 2m/n$. Applying Efron-Stein we get

$$Var(\ell) \leq \frac{1}{2} \cdot \sum_{i=1}^{m} E\left[\left(\ell - \ell'_i\right)^2\right] \leq \frac{m^2}{n}$$

Now we just plug this bound on the variance into Chebyshev inequality and we get the desired inequality. □

Fingerprints. Given a sample S, we define its *fingerprint* F as the set of tuples $(c_a, w(a))$ where for each distinct item a in S, we add to F such a tuple with c_a being equal to the number of copies of a in S. Having a fingerprint of S is sufficient for any algorithm, oblivious of (U, w), to produce a sample S' that is equal to S, up to relabeling of the elements. Since the only allowed queries are testing equality of two items and the weight query, one may easily prove that the execution of the algorithm on these two samples is the same (indeed, these two samples are indistinguishable by the equality and weight queries). Therefore, we can safely assume that an algorithm in the proportional setting takes as an input the fingerprint F of S, rather than S. For algorithms in the hybrid setting, we can assume that it takes as input separately the fingerprint of the proportional and the fingerprint of the uniform samples.

Lemma 15. *Let us have parameters n and $\epsilon < 1/3$. Let $N = n$ with probability $1/2$, and $N = (1-\varepsilon)n$ otherwise. Consider the random instance of the sum estimation problem (U, w) with $|U| = N$ and $w(a) = 1$ for each $a \in U$. Consider a sample of size m and its fingerprint $F_m = \{(c_i, w(a_i)\}_{i=1\dots\ell}$ and define the ratio*

$$\mathcal{R}(F_m) = \frac{P\left(F_m \,|\, N = n\right)}{P\left(F_m \,|\, N = (1-\varepsilon)n\right)}.$$

If $m = o(\sqrt{n}/\varepsilon)$ and $m = o(n)$, then

$$P\left(\frac{98}{100} \le \mathcal{R}(F_m) \le \frac{100}{98}\right) \ge \frac{99}{100}$$

for n large enough.

Proof. We can explicitly compute the likelihood of a given fingerprint $F_m = \{(c_i, w(a_i)\}_{i=1\dots\ell}$ where ℓ is the number of distinct elements as

$$\begin{aligned}P\left(F_m \middle| N = r\right) =& \frac{\ell!}{r^m}\binom{r}{\ell}\binom{m}{c_1 \dots c_\ell} \\ =& \frac{1}{r^{m-\ell}}\prod_{i=1}^{\ell-1}\left(1 - \frac{i}{r}\right)\binom{m}{c_1 \dots c_\ell}\end{aligned}$$

and therefore

$$\begin{aligned}\mathcal{R}(F_m) &= (1-\varepsilon)^{m-\ell} \cdot \prod_{i=1}^{\ell-1} \frac{1 - i/n}{1 - i/(1-\varepsilon)n} \\ &= (1-\varepsilon)^{m-\ell} \cdot \prod_{i=1}^{\ell-1}\left(1 + \frac{\varepsilon i}{(1-\varepsilon)n - i}\right)\end{aligned}$$

Note that $\mathcal{R}(F_m)$ depends only on ℓ. From now on we denote it with $\mathcal{R}(\ell)$.

Now we define an interval $[a, b]$ such that $P(\ell \in [a, b]) \ge 99/100$. To do so, we first compute the expectation of ℓ and then use the concentration bound of Lemma 14. We prove that

$$E\left[\ell | N = n\right] \le E\left[\ell | N = (1-\varepsilon)n\right] \le E\left[\ell | N = n\right] + O\left(\varepsilon \frac{m^2}{n}\right).$$

The expression of $E\left[\ell | N = n\right]$ is given by

$$E\left[\ell | N = n\right] = n \cdot \left(1 - \left(1 - \frac{1}{n}\right)^m\right)$$

since each item is not sampled with probability $(1 - \frac{1}{n})^m$. From this expression, it is apparent that $E\left[\ell | N = n\right]$ is increasing in n. Expanding this formula, we get

$$E\left[\ell | N = (1-\varepsilon)n\right] - E\left[\ell | N = n\right] = \varepsilon \frac{m^2}{2n} + O\left(\varepsilon \frac{m^3}{n^2}\right) = O\left(\varepsilon \frac{m^2}{n}\right)$$

where the last estimate uses the $m = o(n)$ assumption. Now we define $a = E\left[\ell | N = (1-\varepsilon)n\right] - 10m/\sqrt{n}$ and $b = E\left[\ell | N = n\right] + 10m/\sqrt{n}$. Using the last result we proved, we have

$$|b - a| = 20\frac{m}{\sqrt{n}} + O\left(\varepsilon \frac{m^2}{n}\right) = o\left(\frac{1}{\varepsilon}\right).$$

Note that, like all asymptotics in this proof, the $o(1/\epsilon)$ is for the limit $n \to +\infty$ and makes sense even for ϵ being a constant. Thanks to Lemma 14 we have $P(\ell \notin [a,b] \,|\, N = n) \le 1/100$ and $P(\ell \notin [a,b] \,|\, N = (1-\varepsilon)n) \le 1/100$, and therefore $P(\ell \notin [a,b]) \le 1/100$.

Now we give bounds on $\mathcal{R}(a)$ and $\mathcal{R}(b)$. It is apparent from the explicit formula above that $\ell \mapsto \mathcal{R}(\ell)$ is an increasing function. We have

$$\begin{aligned}\mathcal{R}(a) \cdot \frac{99}{100} &\le \mathcal{R}(a) \sum_{k \in [a,b] \cap \mathbb{Z}} P\left(\ell = k \,|\, N = (1-\varepsilon)n\right) \\ &\le \sum_{k \in [a,b] \cap \mathbb{Z}} \mathcal{R}(k) P\left(\ell = k \,|\, N = (1-\varepsilon)n\right) \\ &\le \sum_{k \in [a,b] \cap \mathbb{Z}} P\left(\ell = k \,|\, N = n\right) \le 1\end{aligned}$$

and thus, $\mathcal{R}(a) \le 100/99$. Analogously,

$$\begin{aligned}\frac{1}{\mathcal{R}(b)} \cdot \frac{99}{100} &\le \frac{1}{\mathcal{R}(b)} \sum_{k \in [a,b] \cap \mathbb{Z}} P\left(\ell = k \,|\, N = n\right) \\ &\le \sum_{k \in [a,b] \cap \mathbb{Z}} \frac{1}{\mathcal{R}(k)} P\left(\ell = k \,|\, N = n\right) \\ &\le \sum_{k \in [a,b] \cap \mathbb{Z}} P\left(\ell = k \,|\, N = (1-\varepsilon)n\right) \le 1\end{aligned}$$

and thus, $\mathcal{R}(b) \ge 99/100$. We now have an upper bound on $\mathcal{R}(a)$ and a lower bound on $\mathcal{R}(b)$. However, we need a lower bound on $\mathcal{R}(a)$ and an upper bound on $\mathcal{R}(b)$ (that is, the other way around). For each $k < m$ we have

$$\frac{\mathcal{R}(k+1)}{\mathcal{R}(k)} = \frac{1}{1-\varepsilon} \cdot \left(1 + \frac{\varepsilon k}{(1-\varepsilon)n - k}\right) \le (1 + 2\varepsilon) \cdot \left(1 + \frac{2\varepsilon m}{n}\right) \le 1 + 3\varepsilon$$

for n large enough, where we used $k < m = o(n)$ and $\varepsilon \le 1/3$. Hence,

$$\begin{aligned}\mathcal{R}(b) &\le \mathcal{R}(a) \cdot (1 + 3\varepsilon)^{\lceil |b-a| \rceil} \\ &\le \frac{100}{99} \cdot e^{3\varepsilon \lceil |b-a| \rceil} \\ &\le \frac{100}{99} \cdot e^{o(1)} \le \frac{100}{98}\end{aligned}$$

where the last inequality holds for n large enough because $m = o(\sqrt{n}/\varepsilon)$.

$$\begin{aligned}\mathcal{R}(a) &\ge \mathcal{R}(b) \cdot (1 + 3\varepsilon)^{-\lceil |b-a| \rceil} \\ &\ge \frac{99}{100} \cdot e^{-3\varepsilon \lceil |b-a| \rceil} \\ &\ge \frac{99}{100} \cdot e^{-o(1)} \ge \frac{98}{100}\end{aligned}$$

Finally, we have for n large enough that

$$P\left(\mathcal{R}(\ell) \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) \le P\left(\ell \notin [a,b]\right) \le \frac{1}{100}.$$

$\square$

Using the same approach as Lemma 15, we prove a similar result for the task of estimating the bias p of a Bernoulli random variable up to an additive ε. In this setting, we provide an asymptotic lower bound on the number of samples, where the asymptotics are meant for the limit $(p, \varepsilon) \to 0$.

Lemma 16. *Let $0 < \varepsilon < p$ and set $q = p$ with probability $1/2$, and $q = p - \varepsilon$ otherwise. Let $X_1 \dots X_m$ be a sequence of i.i.d. Bernoulli random variables with bias q, and let $\ell = \sum_{i=1}^m X_i \sim Bin(m, q)$. Define the ratio*

$$\mathcal{R}(\ell) = \frac{P\left(\ell \,|\, q = p\right)}{P\left(\ell \,|\, q = p - \varepsilon\right)}.$$

If $m = o(p/\varepsilon^2)$ then

$$P\left(\frac{98}{100} \leq \mathcal{R}(\ell) \leq \frac{100}{98}\right) \geq \frac{99}{100}.$$

for p (and thus also ϵ) small enough.

Proof. We follow the same scheme we adopted in the proof of Lemma 15. First, we compute $\mathcal{R}(\ell)$ explicitly

$$\mathcal{R}(\ell) = \frac{p^\ell (1-p)^{m-\ell}}{(p-\varepsilon)^\ell (1-p+\varepsilon)^{m-\ell}} = \frac{\left(1 + \frac{\varepsilon}{1-p}\right)^{\ell - m}}{\left(1 - \frac{\varepsilon}{p}\right)^\ell}.$$

We have $E[\ell \,|\, q = p] = mp$, $E[\ell \,|\, q = p - \varepsilon] = m(p - \varepsilon)$, $Var(\ell \,|\, q = p) \leq mp$, and $Var(\ell \,|\, q = p - \varepsilon) \leq mp$. We define $a = m(p - \varepsilon) - 10\sqrt{mp}$ and $b = mp + 10\sqrt{mp}$, and using Chebyshev inequality we have $P(\ell \in [a, b] \,|\, q = p) \geq 99/100$ and $P(\ell \in [a, b] \,|\, q = p - \varepsilon) \geq 99/100$. Hence, $P(\ell \in [a, b]) \geq 99/100$. Notice that $|b - a| = m\varepsilon + 20\sqrt{mp} = o(p/\varepsilon)$ where the second equality holds by the assumption that $m = o(p/\epsilon^2)$. Now, we bound $\mathcal{R}(a)$ and $\mathcal{R}(b)$. Again, we notice that $\ell \mapsto \mathcal{R}(\ell)$ is an increasing function.

$$\begin{aligned}
\mathcal{R}(a) \cdot \frac{99}{100} &\leq \mathcal{R}(a) \sum_{k \in [a,b] \cap \mathbb{Z}} P\left(\ell = k \,|\, q = p - \varepsilon\right) \\
&\leq \sum_{k \in [a,b] \cap \mathbb{Z}} \mathcal{R}(k) P\left(\ell = k \,|\, q = p - \varepsilon\right) \\
&\leq \sum_{k \in [a,b] \cap \mathbb{Z}} P\left(\ell = k \,|\, q = p\right) \leq 1
\end{aligned}$$

and thus, $\mathcal{R}(a) \leq 100/99$. Analogously, we prove $\mathcal{R}(b) \geq 99/100$. For each $k < m$ we have

$$\frac{\mathcal{R}(k+1)}{\mathcal{R}(k)} = \frac{1 + \frac{\varepsilon}{1-p}}{1 - \frac{\varepsilon}{p}} \leq 1 + 3\frac{\varepsilon}{p}$$

for p and ε sufficiently small. Hence,

$$\begin{aligned}
\mathcal{R}(b) &\leq \mathcal{R}(a) \cdot \left(1 + 3\frac{p}{\varepsilon}\right)^{\lceil |b-a| \rceil} \\
&\leq \frac{100}{99} \cdot e^{3\frac{p}{\varepsilon}\lceil |b-a| \rceil} \\
&\leq \frac{100}{99} \cdot e^{o(1)} \leq \frac{100}{98}
\end{aligned}$$

where the last inequality holds for p and ϵ sufficiently small. Analogously, we prove $\mathcal{R}(a) \geq 98/100$. Finally, we have

$$P\left(\mathcal{R}(\ell) \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) \leq P\left(\ell \notin [a,b]\right) \leq \frac{1}{100}.$$

□

4.1 Proportional sampling.

In this section, we assume, as we did in Section 2, that we can sample only proportionally. We prove that $\Omega(\sqrt{n}/\varepsilon)$ samples are necessary to estimate W with probability 2/3, thus PropEstimator is optimal up to a constant factor.

Deciding the number of samples at run-time. In all our lower bounds, we show that it is not possible that an algorithm takes $m = o(T(n,\varepsilon))$ samples in the *worst case* and correctly approximates W with probability 2/3. All these lower bounds safely extend to lower bounds on the expected number of samples $E[m]$. All our proofs work by showing that the Bayes classifier has risk $1/2 - o(1)$. Suppose now that we have an algorithm A that uses in expectation $\mu(n,\epsilon) = o(T(n,\epsilon))$ samples. We now define a classifier as follows. We run A and abort if it uses more than $20\mu(n,\epsilon)$ samples[9]. We return the answer given by A or an arbitrary value if we have aborted the algorithm. By the Markov's inequality, the probability that we abort is at most 1/20. Our classifier has risk $1/3 + 1/20 < 2/5$. Since any constant success probability greater than 1/2 is equivalent up to probability amplification, we also have a classifier with risk 1/3 that uses $O(\mu(n,\epsilon)) = o(T(n,\epsilon)$ samples. Since a Bayes classifier with such parameters does not exist (as we show) and Bayes classifiers are risk-optimal, this is a contradiction.

Theorem 17. *In the proportional setting, there does not exist an algorithm that, for every instance (U,w) with $|U| = n$, takes m samples for $m = o(\sqrt{n}/\varepsilon)$ and returns a $(1 \pm \varepsilon)$-approximation of W with probability 2/3. This holds also when n is known to the algorithm.*

Proof. As already proven, we may assume that the algorithm only gets the fingerprint F_m of the sample S of size m, instead of S itself. In the rest of the proof, we separately consider two cases: $\varepsilon \geq 1/\sqrt{n}$ and $\varepsilon < 1/\sqrt{n}$.

Case $\varepsilon \geq \frac{1}{\sqrt{n}}$: We first define the hard instance (U,w). Define the random variable k as $k = (1-\varepsilon)n$ with probability 1/2 and $k = n$ otherwise, then let $U = \{a_1 \dots a_n\}$ and

$$w(a_i) = \begin{cases} 1 & \text{if } i \leq k \\ 0 & \text{otherwise.} \end{cases}$$

Items with weight zero are never sampled while sampling proportionally while we are sampling uniformly from those with weight 1. Moreover $m = o(\sqrt{n}/\varepsilon) = o(n)$ for $\varepsilon \geq 1/\sqrt{n}$. Hence, this is exactly the settings of Lemma 15. If we let F_m to be the fingerprint of the samples and define $\mathcal{R}(F_m)$ as in Theorem 15, we have

$$P\left(\frac{98}{100} \leq \mathcal{R}(F_m) \leq \frac{100}{98}\right) \geq \frac{99}{100}.$$

Applying Lemma 13 gives us the desired result.

[9]Note that, while the algorithm does not know $\mu(n,\epsilon)$, this is not an issue in this argument. The reason is that a classifier is defined as an arbitrary function from the set of possible samples and private randomness to the set of classes. This allows us to "embed" μ into the classifier

Case $\varepsilon < \frac{1}{\sqrt{n}}$: For convenience, we show an instance of size $n+1$ instead of n. First, we define $s^2 = \min\left\{\frac{\sqrt{n}}{\varepsilon m}, \frac{n}{4}\right\}$ and notice that $s = \omega(1)$ and $s \leq \sqrt{n}/2$. Define the random variable k as $k = n - s\sqrt{n}$ with probability $1/2$ and $k = n$ otherwise. Define the events $\mathcal{E}_1 = \{k = n - s\sqrt{n}\}$ and $\mathcal{E}_2 = \{k = n\}$. We construct (U, w) so that $U = \{a_0 \ldots a_n\}$ and

$$w(a_i) = \begin{cases} \frac{s\sqrt{n}}{\varepsilon} - n & \text{if } i = 0 \\ 1 & \text{if } 1 \leq i \leq k \\ 0 & \text{otherwise.} \end{cases}$$

Notice that our choice of s together with $\varepsilon < 1/\sqrt{n}$ guarantees $n - s\sqrt{n} \geq n/2$ and $s\sqrt{n}/\varepsilon - n = \omega(n)$. We have that $W = \sum_{i=0}^{n} w(a_i)$ differs by more than a factor of $(1+\varepsilon)$ between events $\mathcal{E}_1$ and $\mathcal{E}_2$.

Consider an element $a \in U$ sampled proportionally and define $p_i = P(a \neq a_0 \mid \mathcal{E}_i)$ for $i = 1, 2$. Then,

$$p_2 = \frac{n}{n - s\sqrt{n} + w(a_0)} = \frac{\varepsilon\sqrt{n}}{s}$$
$$p_1 = \frac{n - s\sqrt{n}}{n - s\sqrt{n} + w(a_0)} = \frac{p_2 - \varepsilon}{1 - \varepsilon}$$

and $|p_1 - p_2| \leq \varepsilon$. We perform m samples in total and define the random variable X counting the number of times items other than a_0 are sampled. We define F_X as the fingerprint of the sampled items different from a_0. Given F_X, we may easily reconstruct the fingerprint of the whole sample by adding the tuple $(m - X, w(a_0))$. It thus holds $P(F_m|\mathcal{E}_i) = P(F_X|\mathcal{E}_i)$ for $i \in \{1, 2\}$.

Now we define the event $\mathcal{L} = \{X \leq 30E[X]\}$ and by Markov's inequality $P(\mathcal{L}) \geq 29/30$. Moreover,

$$E[X] = m(p_1 + p_2)/2 \leq m(p_2 + \varepsilon) \leq m(\varepsilon\sqrt{n}/s + \varepsilon) \leq 2\varepsilon m\sqrt{n}/s = o\left(\frac{n}{s}\right).$$

Define $\varepsilon' = s/\sqrt{n}$. Conditioning on $\mathcal{L}$, we have $X \leq 30E[X] = o(\sqrt{n}/\varepsilon') = o(n)$. If we further condition on $X = x$ for some $x \leq 30E[X]$ and we look at F_X, we are exactly in the setting of Lemma 15. Therefore,

$$P\left(\frac{98}{100} \leq \frac{P(F_X \mid X = x, \mathcal{E}_1)}{P(F_X \mid X = x, \mathcal{E}_2)} \leq \frac{100}{98}\right) \geq \frac{99}{100}$$

and integrating over $\mathcal{L}$ we obtain

$$P\left(\frac{98}{100} \leq \frac{P(F_X \mid X, \mathcal{E}_1)}{P(F_X \mid X, \mathcal{E}_2)} \leq \frac{100}{98} \,\middle|\, \mathcal{L}\right) \geq \frac{99}{100}.$$

Now, we will bound the ratio

$$\mathcal{R}(X) = \frac{P(X \mid \mathcal{E}_1)}{P(X \mid \mathcal{E}_2)}.$$

We have $X = \sum_{i=1}^{m} X_i$, where X_i is an indicator for the i-th sample not being equal to a_0. Therefore, $X \mid \mathcal{E}_j \sim Bin(m, p_j)$ for $j = 1, 2$. It holds, $|p_1 - p_2| \leq \varepsilon$. Because $m = o(\sqrt{n}/\epsilon)$ and by the definition of s, we have $m = o(\sqrt{n}/(s\varepsilon)) = o(p_2/\varepsilon^2)$. We can apply Theorem 16 and obtain

$$P\left(\frac{98}{100} \leq \mathcal{R}(X) \leq \frac{100}{98}\right) \geq \frac{99}{100}.$$

Finally, we put the bounds together. We consider the ratio

$$\mathcal{R}(F_m) = \frac{P(F_m \mid \mathcal{E}_1)}{P(F_m \mid \mathcal{E}_2)} = \frac{P(F_X \mid \mathcal{E}_1)}{P(F_X \mid \mathcal{E}_2)} = \frac{P(X \mid \mathcal{E}_1) \cdot P(F_X \mid X, \mathcal{E}_1)}{P(X \mid \mathcal{E}_2) \cdot P(F_X \mid X, \mathcal{E}_2)}.$$

By the union bound, along with $7/8 < (98/100)^2$, we get

$$P\left(\mathcal{R}(F_m) \notin \left[\frac{7}{8}, \frac{8}{7}\right]\right) \leq$$

$$P(\bar{\mathcal{L}}) + P\left(\frac{P(F_X \mid X, \mathcal{E}_1)}{P(F_X \mid X, \mathcal{E}_2)} \notin \left[\frac{98}{100}, \frac{100}{98}\right] \,\middle|\, \mathcal{L}\right) + P\left(\frac{P(X \mid \mathcal{E}_1)}{P(X \mid \mathcal{E}_2)} \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) \leq$$

$$\frac{1}{30} + \frac{1}{100} + \frac{1}{100} \leq \frac{1}{15}.$$

We can apply Theorem 13 and conclude the proof. □

4.2 Sum estimation in hybrid setting, known n.

In this section, we assume, as we did in Section 3, that we can sample both proportionally and uniformly. We will prove that $\Omega(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n))$ samples are necessary to estimate W with probability 2/3. This complexity is the minimum of the sample complexity of the HybridEstimator and (up to a logarithmic factor) the complexity of the standard coupon collector algorithm.

Theorem 18. *In the hybrid setting, there does not exist an algorithm that, for every instance (U, w) with $|U| = n$, takes $m = o(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n))$ proportional and uniform samples and returns a $(1 \pm \varepsilon)$-approximation of W with probability 2/3. This holds also when n is known to the algorithm.*

Proof. It is enough to prove that for $\varepsilon \geq 8/\sqrt{n}$, any algorithm returning a $(1 \pm \varepsilon)$-approximation of W with probability 2/3 must take $\Omega(\sqrt[3]{n}/\varepsilon^{4/3})$ samples. Indeed, if $\varepsilon < 8/\sqrt{n}$, a $(1 \pm \varepsilon)$-approximation is also a $(1 \pm 8/\sqrt{n})$-approximation, and then $\Omega(n)$ samples are necessary. In either case we then need $\Omega(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n))$ samples.

Define the random variable k as $k = (1 - \varepsilon)n^{2/3}/\varepsilon^{2/3}$ with probability 1/2 and $k = n^{2/3}/\varepsilon^{2/3}$ otherwise. Define the events $\mathcal{E}_1 = \{k = (1 - \varepsilon)n^{2/3}/\varepsilon^{2/3}\}$ and $\mathcal{E}_2 = \{k = n^{2/3}/\varepsilon^{2/3}\}$. We construct (U, w) so that $U = \{a_1 \dots a_n\}$ and

$$w(a_i) = \begin{cases} 1 & \text{if } 1 \leq i \leq k \\ 0 & \text{otherwise.} \end{cases}$$

We have that $W = \sum_{i=1}^{n} w(a_i)$ differs by more than a factor of $(1 + \varepsilon)$ between events $\mathcal{E}_1$ and $\mathcal{E}_2$. Notice that $k \leq n/4$ as $\varepsilon \geq 8/\sqrt{n}$. Let S_p be the multiset of proportional samples and S_u the multiset of uniform samples. Let $T_h = (S_p \cup S_u) \cap \{k+1, \cdots, n\}$ and $T_l = (S_p \cup S_u) \cap \{1, \cdots, k\}$. Let $F_l = \{(c_i, w(a_i))\}_i$ be the fingerprint of T_l and $F_h = \{(c_i, w(a_i))\}_i$ of be the fingerprint of T_h. We now argue hat (F_l, F_h) is sufficient for any algorithm, oblivious of the choice of k, to reconstruct sample multisets S'_u and S'_p distributed as S_u and S_p. We pick $|F_l| - m$ items at random from F_l and let S'_u be the multiset of these items, together with all items in F_h. We let S'_p be the multiset of the items left in F_l. It is easy to verify that $(S'_u, S'_p) \sim (S_u, S_p)$. Thus, we can assume that the algorithm is given (F_l, F_h) as input, instead of the sample multisets S_u and S_p.

Consider $a \in U$ sampled uniformly, and define $p_i = P(w(a) = 1 \mid \mathcal{E}_i)$ for $i = 1, 2$. Then,

$$p_1 = \frac{n^{2/3}/\varepsilon^{2/3}}{n} = \frac{1}{n^{1/3}\varepsilon^{2/3}}$$

$$p_2 = \frac{(1-\varepsilon)n^{2/3}/\varepsilon^{2/3}}{n} = \frac{1}{n^{1/3}\varepsilon^{2/3}} - \frac{\varepsilon^{1/3}}{n^{1/3}}$$

and defining $\varepsilon' = \varepsilon^{1/3}/n^{1/3}$ we obtain $p_2 = p_1 - \varepsilon'$. Let $X = |S_u \cap \{1, \cdots, k\}|$. Since $X \,|\, \mathcal{E}_j \sim Bin(m, p_j)$ for $j = 1, 2$ and $m = o(n^{1/3}/\varepsilon^{4/3}) = o(p_1/\varepsilon'^2)$, we can apply Theorem 16 and obtain

$$P\left(\frac{98}{100} \leq \frac{P\left(X \,|\, \mathcal{E}_1\right)}{P\left(X \,|\, \mathcal{E}_2\right)} \leq \frac{100}{98}\right) \geq \frac{99}{100}.$$

Conditioning on $X = x$ for some $x = 1 \dots m$, F_l represents a sample of $x + |S_p|$ items drawn uniformly from a set of cardinality k, so we are in the setting of theorem 15. Moreover, we have

$$|F_l| \leq |S_p| + |S_u| = o\left(\frac{\sqrt[3]{n}}{\varepsilon^{4/3}}\right) = o\left(\frac{\sqrt{n^{2/3}/\varepsilon^{2/3}}}{\varepsilon}\right).$$

Hence, Theorem 15 holds and we have

$$P\left(\frac{98}{100} \leq \frac{P\left(F_l \,|\, X = x, \mathcal{E}_1\right)}{P\left(F_l \,|\, X = x, \mathcal{E}_2\right)} \leq \frac{100}{98}\right) \geq \frac{99}{100}$$

and integrating over $x = 1 \dots m$ we have

$$P\left(\frac{98}{100} \leq \frac{P\left(F_l \,|\, X, \mathcal{E}_1\right)}{P\left(F_l \,|\, X, \mathcal{E}_2\right)} \leq \frac{100}{98}\right) \geq \frac{99}{100}.$$

Similarly, we have that $|F_h| \leq |S_u| = o(\sqrt[3]{n}/\varepsilon^{4/3}) = o(\sqrt{n}/\varepsilon)$ where the second inequality holds because we are assuming $\varepsilon > 8/\sqrt{n}$. Moreover, conditioning on $X = x$ for some $x = 1 \dots m$, F_h represent a sample of $|S_u| - x$ items drawn uniformly from a set of size $n - k$. It holds $n - k \geq 3n/4$, and $n - k$ thus differs by at most a factor $1 - \varepsilon$ between the two events $\mathcal{E}_1, \mathcal{E}_2$. Again, we are in the right setting to apply Theorem 15, and integrating over $x = 1 \dots m$ gives

$$P\left(\frac{98}{100} \leq \frac{P\left(F_h \,|\, X, \mathcal{E}_1\right)}{P\left(F_h \,|\, X, \mathcal{E}_2\right)} \leq \frac{100}{98}\right) \geq \frac{99}{100}.$$

We are now ready to put everything together. Note that F_l and F_h are independent once conditioned on $(\mathcal{E}_1, X)$ or $(\mathcal{E}_2, X)$. Define the ratio

$$\mathcal{R}(F_l, F_h) = \frac{P\left((F_l, F_h) \,|\, \mathcal{E}_1\right)}{P\left((F_l, F_h) \,|\, \mathcal{E}_2\right)} = \frac{P\left(X \,|\, \mathcal{E}_1\right) \cdot P\left(F_l \,|\, X, \mathcal{E}_1\right) \cdot P\left(F_h \,|\, X, \mathcal{E}_1\right)}{P\left(X \,|\, \mathcal{E}_2\right) \cdot P\left(F_l \,|\, X, \mathcal{E}_2\right) \cdot P\left(F_h \,|\, X, \mathcal{E}_2\right)}$$

Using the union bound, along with $7/8 < (98/100)^3$, we get

$$\begin{aligned} &P\left(\mathcal{R}(F_l, F_h) \notin \left[\frac{7}{8}, \frac{8}{7}\right]\right) \leq \\ &P\left(\frac{P\left(X \,|\, \mathcal{E}_1\right)}{P\left(X \,|\, \mathcal{E}_2\right)} \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) + P\left(\frac{P\left(F_l \,|\, X, \mathcal{E}_1\right)}{P\left(F_l \,|\, X, \mathcal{E}_2\right)} \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) + \\ &P\left(\frac{P\left(F_h \,|\, X, \mathcal{E}_1\right)}{P\left(F_h \,|\, X, \mathcal{E}_2\right)} \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) \leq \\ &\frac{1}{100} + \frac{1}{100} + \frac{1}{100} < \frac{1}{15}. \end{aligned}$$

We can apply Theorem 13 and conclude the proof.

□

4.3 Sum estimation in hybrid setting, unknown n.

We now prove a lower bound for the hybrid setting, in case the algorithm does not know n.

Theorem 19. *In the hybrid setting, there does not exist an algorithm that, for every instance (U, w), takes $m = o(\min(\sqrt{n}/\varepsilon, n))$ samples and returns a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$. This holds also when the algorithm is provided with an advice $\tilde{n}$ such that $(1-\varepsilon)n \le \tilde{n} \le n$.*

Proof. Employing the same argument as in Theorem 18, it is sufficient to prove that for $\varepsilon \ge 1/\sqrt{n}$ a lower bound of $\Omega(\sqrt{n}/\varepsilon)$ holds.

Consider the instance (U, w) where $w(a) = 1$ for each $a \in U$ and we set $|U| = n$ with probability $1/2$ and $|U| = (1 - \varepsilon)n$ otherwise. Providing a $(1 \pm \varepsilon)$-approximation of W is equivalent to distinguishing between the two cases. On this instance, sampling uniformly and proportionally is the same. Therefore, we are in the setting of Theorem 15. Combining Theorem 15 and Theorem 13 like in the proofs above, we get that no classifier can distinguish between $|U| = n$ and $|U| = (1-\varepsilon)n$ with probability $2/3$ using $o(\sqrt{n}/\varepsilon)$ samples. $\square$

5 Counting Edges in a Graph.

In this section, we show an algorithm that estimates the average degree of a graph $G = (V, E)$ in the model in which we are allowed to perform random vertex queries, random edge queries, and degree queries. Recall that a *random vertex query* returns a uniform sample form V, a *random edge queries* returns a uniform sample from E, and a *degree queries* returns $deg(v)$ when we provide $v \in V$ as argument. In this section, we denote the number of vertices and edges with n and m respectively.

Here, we show that HARMONICESTIMATOR from Section 3, can be adapted to estimate the average degree d. It achieves a complexity of $O(\frac{m \log\log n}{n'\varepsilon^2} + \frac{n}{n'\varepsilon^2})$ in expectation, where n' is the number of non-isolated[10] vertices. This is very efficient when there are few isolated vertices and the graph is sparse. Moreover, the only way we use sampling of vertices is to estimate the number of non-isolated vertices. Therefore, if we assume that there are no isolated vertices in the graph, it is sufficient to only be able to uniformly sample edges.

Our approach is similar to that of [10] but the authors in the paper do not prove bounds on the time complexity. Moreover, their algorithm only works when there are no isolated vertices.

Theorem 20. *Given a graph $G = (V, E)$, consider a model that allows (1) random vertex queries, (2) random edge queries, and (3) degree queries. In this model, there exists an algorithm that, with probability at least $2/3$, returns a $(1 \pm \varepsilon)$-approximation $\hat{d}$ of the average degree $d = 2m/n$. This algorithm performs $O(\frac{m \log\log d}{n'\varepsilon^2} + \frac{n}{n'\varepsilon^2})$ queries in expectation, where n' is the number of non-isolated vertices.*

Proof. We first show an algorithm that is given $\tilde{\theta}$ such that $d \le \tilde{\theta}$, has time complexity $O(\frac{\tilde{\theta} n}{\varepsilon^2 n'} + \frac{n}{\varepsilon^2 n'})$, and is correct with probability $2/3$. We define a sum estimation problem (U, w). We set the universe to be $U = V$ and for each vertex $v \in U$, we set its weight $w(v) = deg(v)$. Then $W = \sum_{a \in U} w(a) = 2m$ and $W/n = d$. Sampling an edge uniformly at random and picking one of its endpoints at random corresponds to sampling a vertex proportionally to its weight. Moreover, we can sample vertices uniformly. Therefore, we are able to implement both queries of the hybrid setting. We run HARMONICESTIMATOR$(\varepsilon, \tilde{\theta}, 1)$. By Theorem 9, it returns with probability at least $2/3$ a $1 \pm \varepsilon$-approximation of d, and its sample complexity is $O(\frac{\tilde{\theta} n}{\varepsilon^2 n'} + \frac{n}{\varepsilon^2 n'})$ since what is called p in Theorem 9 is now n'/n.

[10]Recall that a vertex is isolated if it has degree 0.

It remains to get rid of the need for advice $\tilde{\theta}$. We use the standard technique of performing a geometric search. See, for example, [8] for more details. We initialize $\tilde{\theta} = 1$ and in each subsequent iteration, we double $\tilde{\theta}$. Let K be a large enough constant. In each iteration, we run $K \log\log\tilde{\theta}$ independent copies of HarmonicEstimator$(\varepsilon, \tilde{\theta}, 1)$ and denote with $d_1 \dots d_{K\log\log\tilde{\theta}}$ the returned estimates. We define $\hat{d}$ as the median of $d_1 \dots d_{K\log\log\tilde{\theta}}$. We say that the d_i succeeds if both the following hold: (i) $d_i \geq d/20$, (ii) $\tilde{\theta} < d$ or d_i is a $(1 \pm \varepsilon)$-approximation of d. Otherwise we say that d_i fails. We extend this definition to $\hat{d}$. Thanks to Theorem 9, d_i fails with probability $\leq 1/3 + 1/20$. By a standard argument based on the Chernoff bound, for K large enough, we have that $\hat{d}$ fails with probability at most $2/(\pi \log^2(2\tilde{\theta}))$. Denote with $\mathcal{E}_j$ the event that $\hat{d}$ succeeds at iteration j (i.e., when $\tilde{\theta} = 2^{j-1}$). Define $\mathcal{E} = \bigcap_{j\geq 0} \mathcal{E}_j$. Union bound gives $P(\mathcal{E}) \geq 1 - \frac{2}{\pi}\sum_{j>0}\frac{1}{j^2} = 2/3$. We stop our algorithm when $\hat{d} \leq \tilde{\theta}/20$ and return $\hat{d}$. Conditioning on $\{\hat{d} \leq \tilde{\theta}/20\} \cap \mathcal{E}$, we have $\tilde{\theta}/20 \geq \hat{d} \geq d/20$. This implies that $\tilde{\theta} \geq d$ and hence $\hat{d}$ is a $(1 \pm \varepsilon)$-approximation of d. Since $P(\mathcal{E}) \geq 2/3$, we have that with probability $2/3$ the returned value is a $(1 \pm \varepsilon)$-approximation of d.

One iteration of our algorithm has time complexity $O(\frac{\tilde{\theta} n \log\log\tilde{\theta}}{\varepsilon^2 n'})$. We argue that the expected complexity is dominated by the first iteration in which $\tilde{\theta} \geq 40\,d$. The time complexity of each additional iteration (conditioned on being executed) increases by a factor $2 + o(1)$. Each additional iteration is executed only if the previous one resulted in an estimate $\hat{d} > \tilde{\theta}/20 \geq 2\,d$. This happens only when $\hat{d}$ is not a $(1 \pm \varepsilon)$-approximation of d, and assuming a correct advice $\tilde{\theta} \geq d$ this happens outside of $\mathcal{E}$. Therefore, we execute each additional iteration with probability $\leq 1/3$. Since the time spend in each iteration increases by a factor $2 + o(1)$ while the probability of executing the iteration decreases by a factor of 3, the expected complexity contributed by each additional iteration for $\tilde{\theta} \geq 40\,d$ decreases by a factor of $2/3 + o(1)$. Therefore, the expected complexity is dominated (up to a constant factor) by the first execution with $\tilde{\theta} \geq 40\,d$. If $d \geq 1$, then in this iteration, we have $\tilde{\theta} = \Theta(d)$. The time complexity is then $O(\frac{m \log\log d}{\epsilon^2 n'})$. If $d < 1$, then it holds $\theta = O(1)$ in this execution. The complexity is then $O(\frac{n}{n'\epsilon^2})$. This gives total time complexity of $O(\frac{m \log\log d}{n'\epsilon^2} + \frac{n}{n'\epsilon^2})$. □

6 Open Problems.

We believe there are many interesting open problems related to our work. We now give a non-comprehensive list of questions that we think would give more understanding of weighted sampling and its applications.

More efficient algorithm for spacial classes of inputs. Are there some large classes of inputs for which it is possible to get a more efficient algorithm? Can the problem be parameterized by some additional parameters apart from n, ε (e.g. empirical variance) that tend to be small in practice?

Different sampling probabilities. Are there settings where one may efficiently sample with probability depending on the value of an item but not exactly proportional? Could this be used to give a general algorithm for estimating the sum W? An example of such a result is [4] where the authors show an efficient algorithm for estimating the average degree of a graph when sampling vertices with probabilities proportional to $\frac{m}{n} + d(v)$.

Get a complete understanding of edge counting. The complexity of the problem of approximately counting edges in a graph is understood in terms of n, m in the setting where we can only sample vertices uniformly at random. What is the complexity of counting edges when we allow only random edge queries? What if both random edge and random vertex queries are allowed? As we show, it may be useful to parameterize the problem by the fraction of vertices that are not isolated. What is the complexity of the problem under such parameterization?

Acknoweldgements.

We are grateful to Rasmus Pagh for his advice. We would like to thank our supervisor, Mikkel Thorup, for helpful discussions and his support. We would like to thank Talya Eden for her advice regarding related work.

References

[1] M. Aliakbarpour, A. S. Biswas, T. Gouleakis, J. Peebles, R. Rubinfeld, and A. Yodpinyanee, *Sublinear-Time Algorithms for Counting Star Subgraphs via Edge Sampling*, Algorithmica, 80 (2018), pp. 668–697.

[2] S. Assadi, M. Kapralov, and S. Khanna, *A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling*, Leibniz International Proceedings in Informatics, LIPIcs, 124 (2019).

[3] C. Canonne and R. Rubinfeld, *Testing probability distributions underlying aggregated data*, in Automata, Languages, and Programming, J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, eds., Berlin, Heidelberg, 2014, Springer Berlin Heidelberg, pp. 283–295.

[4] A. Dasgupta, R. Kumar, and T. Sarlos, *On estimating the average degree*, in Proceedings of the 23rd International Conference on World Wide Web, WWW '14, New York, NY, USA, 2014, Association for Computing Machinery, p. 795–806.

[5] T. Eden, D. Ron, and C. Seshadhri, *Sublinear time estimation of degree distribution moments: The degeneracy connection*, in Leibniz International Proceedings in Informatics, LIPIcs, vol. 80, Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, jul 2017.

[6] U. Feige, *On sums of independent random variables with unbounded variance and estimating the average degree in a graph*, SIAM J. Comput., 35 (2006), pp. 964–984.

[7] H. Fichtenberger, M. Gao, and P. Peng, *Sampling Arbitrary Subgraphs Exactly Uniformly in Sublinear Time*, in 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020), A. Czumaj, A. Dawar, and E. Merelli, eds., vol. 168 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2020, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 45:1–45:13.

[8] O. Goldreich and D. Ron, *Approximating average parameters of graphs*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, J. Díaz, K. Jansen, J. D. P. Rolim, and U. Zwick, eds., Berlin, Heidelberg, 2006, Springer Berlin Heidelberg, pp. 363–374.

[9] D. G. Horvitz and D. J. Thompson, *A generalization of sampling without replacement from a finite universe*, Journal of the American Statistical Association, 47 (1952), pp. 663–685.

[10] L. Katzir, E. Liberty, and O. Somekh, *Estimating sizes of social networks via biased sampling*, in Proceedings of the 20th International Conference on World Wide Web, WWW '11, New York, NY, USA, 2011, Association for Computing Machinery, p. 597–606.

[11] R. J. Lipton, J. F. Naughton, D. A. Schneider, and S. Seshadri, *Efficient sampling strategies for relational database operations*, Theoretical Computer Science, 116 (1993), pp. 195–226.

[12] R. Motwani, R. Panigrahy, and Y. Xu, *Estimating sum by weighted sampling*, in Proceedings of the 34th International Conference on Automata, Languages and Programming, ICALP'07, Berlin, Heidelberg, 2007, Springer-Verlag, p. 53–64.

[13] K. Onak and X. Sun, *Probability-revealing samples*, in AISTATS, 2018.

[14] C. Seshadhri, *A simpler sublinear algorithm for approximating the triangle count*, CoRR, (2015).

[15] A. Shankha Biswas, T. Eden, and R. Rubinfeld, *Towards a Decomposition-Optimal Algorithm for Counting and Sampling Arbitrary Motifs in Sublinear Time*, arXiv e-prints, (2021), p. arXiv:2107.06582.

[16] J. Tětek and M. Thorup, *Sampling and counting edges via vertex accesses*, arXiv e-prints, (2021), p. arXiv:2107.03821.

[17] O. Watanabe, *Sequential sampling techniques for algorithmic learning theory*, Theoretical Computer Science, 348 (2005), pp. 3–14. Algorithmic Learning Theory (ALT 2000).

Appendix C

SoCG: Online Packing to Minimize Area or Perimeter

Online Packing to Minimize Area or Perimeter

Mikkel Abrahamsen* Lorenzo Beretta*

January 21, 2021

Abstract

We consider online packing problems where we get a stream of axis-parallel rectangles. The rectangles have to be placed in the plane without overlapping, and each rectangle must be placed without knowing the subsequent rectangles. The goal is to minimize the perimeter or the area of the axis-parallel bounding box of the rectangles. We either allow rotations by 90° or translations only.

For the perimeter version we give algorithms with an absolute competitive ratio slightly less than 4 when only translations are allowed and when rotations are also allowed.

We then turn our attention to minimizing the area and show that the asymptotic competitive ratio of any algorithm is at least $\Omega(\sqrt{n})$, where n is the number of rectangles in the stream, and this holds with and without rotations. We then present algorithms that match this bound in both cases and the competitive ratio is thus optimal to within a constant factor. We also show that the competitive ratio cannot be bounded as a function of $\textsc{Opt}$. We then consider two special cases.

The first is when all the given rectangles have aspect ratios bounded by some constant. The particular variant where all the rectangles are squares and we want to minimize the area of the bounding square has been studied before and an algorithm with a competitive ratio of 8 has been given [Fekete and Hoffmann, Algorithmica, 2017]. We improve the analysis of the algorithm and show that the ratio is at most 6, which is tight.

The second special case is when all edges have length at least 1. Here, the $\Omega(\sqrt{n})$ lower bound still holds, and we turn our attention to lower bounds depending on $\textsc{Opt}$. We show that any algorithm for the translational case has an asymptotic competitive ratio of at least $\Omega(\sqrt{\textsc{Opt}})$. If rotations are allowed, we show a lower bound of $\Omega(\sqrt[4]{\textsc{Opt}})$. For both versions, we give algorithms that match the respective lower bounds: With translations only, this is just the algorithm from the general case with competitive ratio $O(\sqrt{n}) = O(\sqrt{\textsc{Opt}})$. If rotations are allowed, we give an algorithm with competitive ratio $O(\min\{\sqrt{n}, \sqrt[4]{\textsc{Opt}}\})$, thus matching both lower bounds simultaneously.

1 Introduction

Problems related to packing appear in a plethora of big industries. For instance, two-dimensional versions of packing arise when a given set of pieces have to be cut out from a large piece of material so as to minimize waste. This is relevant to clothing production where cutting patterns are cut out from a roll of fabric, and similarly in leather, glass, wood, and sheet metal cutting.

In some applications, it is important that the pieces are placed in an *online* fashion. This means that the pieces arrive one by one and we need to decide the placement of one piece before we know the ones that will come in the future. This is in contrast to *offline* problems, where all the pieces are known in advance. Problems related to packing were some of the first for which online algorithms were described and analyzed. Indeed, the first use of the terms "online" and "offline" in the context of approximation algorithms was in the early 1970s and used for algorithms for bin-packing problems [14].

In this paper, we study online packing problems where the pieces can be placed anywhere in the plane as long as they do not overlap. The goal is to minimize the region occupied by the pieces. The pieces are axis-parallel rectangles, and they may or may not be rotated by 90°. We want to minimize the size of the axis-parallel bounding box of the pieces, and the size of the box is either the perimeter or the area. This results in four problems: PerimeterRotation, PerimeterTranslation, AreaRotation, and AreaTranslation.

*Basic Algorithms Research Copenhagen (BARC), University of Copenhagen. BARC is supported by the VILLUM Foundation grant 16582. Lorenzo Beretta received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 801199.

Competitive analysis The *competitive ratio* of an online algorithm is the equivalent of the *approximation ratio* of an (offline) approximation algorithm. The usual definitions [7, 9, 11] of competitive ratio (or *worst case ratio*, as it may also be called [11]) can only be used to describe that the cost of the solution produced by an online algorithm is at most some constant factor higher than the cost Opt of the optimal (offline) solution. In the study of approximation algorithms, it is often the case that the approximation ratio is described not just as a constant, but as a more general function of the input. In the same way, we generalize the definition of competitive ratios to support such statements about online algorithms.

Consider an algorithm A for one of the packing problems studied in this paper. Let $\mathcal{L}$ be the set of non-empty streams of rectangular pieces. For a stream $L \in \mathcal{L}$, we define $A(L)$ to be the cost of the packing produced by A and let $\textsc{Opt}(L)$ be the cost of the optimal (offline) packing. We say that A has an *absolute competitive ratio* of $f(L)$, for some function $f : \mathcal{L} \longrightarrow \mathbb{R}^+$ which may just be a constant, if

$$\sup_{L\in\mathcal{L}} \frac{A(L)}{\textsc{Opt}(L)f(L)} \leq 1.$$

We say that A has an *asymptotic competitive ratio* of $f(L)$ if

$$\limsup_{c\longrightarrow\infty} \left(\sup\left\{\frac{A(L)}{\textsc{Opt}(L)f(L)} \mid L \in \mathcal{L} \text{ and } \textsc{Opt}(L) = c\right\}\right) \leq 1.$$

In this paper, the functions $f(L)$ that we consider will be (i) constants, (ii) functions of the number of pieces $n = |L|$, (iii) functions of $\textsc{Opt}(L)$.

By definition, if A has an absolute competitive ratio of $f(L)$, then A also has an asymptotic competitive ratio of $f(L)$, but A may also have a smaller asymptotic competitive ratio $g(L) < f(L)$. However, the following easy lemma shows that for the problems studied in this paper, any constant asymptotic competitive ratio can be matched to within an arbitrarily small difference by an absolute competitive ratio.

Lemma 1. *For the problems studied in this paper, if an algorithm A has an asymptotic competitive ratio of some constant $c > 1$, then for every $\varepsilon > 0$, there is an algorithm A' with absolute competitive ratio $c+\varepsilon$. It follows that any constant lower bound on the absolute competitive ratio is also a lower bound on the asymptotic competitive ratio.*

Proof. Let $n > 0$ be so large that when $\textsc{Opt}(L) \geq n$, we have $\frac{A(L)}{c\textsc{Opt}(L)} \leq 1 + \varepsilon/c$. When the first piece p of a stream L is given, A' chooses a scale factor $\lambda > 0$ big enough that when p is scaled up by λ, the resulting piece $p' := \lambda p$ alone has cost n (i.e., the area or the perimeter of p' is n). The algorithm A' now imitates the strategy of A on the stream λL we get by scaling up all pieces of L by λ. We then get that

$$\frac{A'(L)}{(c+\varepsilon)\textsc{Opt}(L)} = \frac{A(\lambda L)}{(c+\varepsilon)\textsc{Opt}(\lambda L)} \leq \frac{(1+\varepsilon/c)c}{c+\varepsilon} = 1. \qquad \square$$

For this reason, we do not distinguish between absolute and asymptotic competitive ratios when the ratio is a constant. Note that the argument does not work when the competitive ratio is a non-constant function of Opt.

Results and structure of the paper We develop online algorithms for the perimeter versions PerimeterRotation and PerimeterTranslation, both with a competitive ratio slightly less than 4. These algorithms are described in Section 2. The idea is to partition the positive quadrant into *bricks*, which are axis-parallel rectangles with aspect ratio $\sqrt{2}$. In each brick, we build a stack of pieces which would be too large to place in a brick of smaller size. Online packing algorithms using higher-dimensional bricks were described by Januszewski and Lassak [15] and our algorithms are inspired by an algorithm of Fekete and Hoffmann [13] that we will get back to. Interestingly, we show in Section 2.2 that a more direct adaptation of the algorithm of Fekete and Hoffmann has a competitive ratio of at least 4, and is thus inferior to the algorithm we describe. We also give a lower bound of 4/3 for the version with translations and 5/4 for the version with rotations.

In Section 3, we study the area versions AreaRotation and AreaTranslation. We show in Section 3.1 that for any algorithm A processing a stream of n pieces cannot achieve a better competitive

Measure	**Version**	**Trans./Rot.**	**Lower bound**	**Upper bound**
Perimeter	General	Translation	4/3, Sec. 2.3	$4-\varepsilon$, Sec. 2.1
		Rotation	5/4, Sec. 2.3	$4-\varepsilon$, Sec. 2.1
Area	General	Translation	$\Omega(\sqrt{n})$ & $\forall f: \Omega(f(\text{Opt}))$, Sec. 3.1	$O(\sqrt{n})$, Sec. 3.2
		Rotation	$\Omega(\sqrt{n})$ & $\forall f: \Omega(f(\text{Opt}))$, Sec. 3.1	$O(\sqrt{n})$, Sec. 3.2
	Sq.-in-sq.	N/A	16/9, Sec. 3.3	6, Sec. 3.3
	Long edges	Translation	$\Omega(\sqrt{\text{Opt}})$, Sec. 3.4	$O(\sqrt{n}) = O(\sqrt{\text{Opt}})$, Sec. 3.5
		Rotation	$\Omega(\max\{\sqrt{n}, \sqrt[4]{\text{Opt}}\})$, Sec. 3.1 and 3.4	$O(\min\{\sqrt{n}, \sqrt[4]{\text{Opt}}\})$, Sec. 3.5

Table 1: Results of this paper.

ratio than $\Omega(\sqrt{n})$, and this holds for all online algorithms and with and without rotations allowed. It also holds in the special case where all the edges of pieces have length at least 1. We furthermore show that when the pieces can be arbitrary, there can be given no bound on the competitive ratio as a function of Opt for AreaRotation nor AreaTranslation. In Section 3.2 we describe the algorithms DynBoxTrans and DynBoxRot, which achieve a $O(\sqrt{n})$ competitive ratio for AreaTranslation and AreaRotation, respectively, for an arbitrary stream of n pieces. This is thus optimal up to a constant factor when measuring the competitive ratio as a function of n. Both algorithms use a row of boxes of exponentially increasing width and dynamically adjusted height. In these boxes, we pack pieces using a next-fit shelf algorithm, which is a classic online strip packing algorithm first described by Baker and Schwartz [6].

We then turn our attention to two special cases.

The first special case is when the aspect ratio is bounded by a constant $\alpha \geq 1$. A case of particular interest is when all pieces are squares, i.e., $\alpha = 1$. It is natural to have the same requirement to the container as to the pieces, so let us assume that the goal is to minimize the area of the axis-parallel bounding square of the pieces, and call the problem SquareInSquareArea. This problem was studied by Fekete and Hoffmann [13], and they gave an algorithm for the problem and proved that it was 8-competitive. We prove that the same algorithm is in fact 6-competitive and that this is tight. It easily follows that if the aspect ratio is bounded by an arbitrary constant $\alpha \geq 1$ or if the goal is to minimize the area of the axis-parallel bounding rectangle, we also get a $O(1)$-competitive algorithm.

The second special case is when all edges are *long*, that is, when they have length at least 1 (any other constant will work too). In Section 3.4, we show that under this assumption, there is a lower bound of $\Omega(\sqrt{\text{Opt}})$ for the asymptotic competitive ratio of AreaTranslation, whereas for AreaRotation, we get the lower bound $\Omega(\sqrt[4]{\text{Opt}})$. In Section 3.5, we provide algorithms for the area versions when the edges are long. For both problems AreaRotation and AreaTranslation, we give algorithms that match the lower bounds of Section 3.4 to within a constant factor. With translations only, this is just the algorithm from the general case with competitive ratio $O(\sqrt{n}) = O(\sqrt{\text{Opt}})$. The algorithm with ratio $O(\sqrt[4]{\text{Opt}})$ for the rotational case follows the same scheme as the algorithms for arbitrary rectangles of Section 3.2, but differ in the way we dynamically increase boxes' heights. We finally describe an algorithm for the rotational case with competitive ratio $O(\min\{\sqrt{n}, \sqrt[4]{\text{Opt}}\})$, thus matching the lower bounds $\Omega(\sqrt{n})$ and $\Omega(\sqrt[4]{\text{Opt}})$ simultaneously. Actually, the two lower bounds for AreaRotation can be summarized by $\Omega(\max\{\sqrt{n}, \sqrt[4]{\text{Opt}}\})$, while we manage to achieve a competitive ratio of $O(\min\{\sqrt{n}, \sqrt[4]{\text{Opt}}\})$. However, this gives no contradiction, it simply proves that the *edge cases* that have a competitive ratio of at least $\Omega(\sqrt[4]{\text{Opt}})$ must satisfy $\text{Opt} = O(n^2)$, and those for which the competitive ratio is at least $\Omega(\sqrt{n})$ satisfy $n = O(\sqrt{\text{Opt}})$.

We summarize the results in Table 1.

Related work The literature on online packing problems is rich. See the surveys of Christensen, Khan, Pokutta, and Tetali [9], van Stee [25, 26], and Csirik and Woeginger [11] for an overview. It seems that the vast majority of previous work on online versions of two-dimensional packing problems is concerned with either bin packing (packing the pieces into a minimum number of unit squares) or strip packing (packing the pieces into a strip of unit width so as to minimize the total height of the pieces). From a mathematical point of view, we find the problems studied in this paper perhaps even more fundamental than these important problems in the sense that we give no restrictions on where to place the pieces, whereas the pieces are restricted by the boundaries of the bins and the strip in bin and strip packing.

Another related problem is to find the critical density of online packing squares into a square. In other words, what is the maximum $\Sigma \leq 1$ such that there is an online algorithm that packs any stream of squares of total area at most Σ into the unit square? This was studied, among others, by Fekete and Hoffmann [13] and Brubach [8]. Lassak [16] and Januszewski and Lassak [15] studied higher-dimensional versions of this problem.

Milenkovich [20] studied generalized offline versions of the minimum area problem: Translate k given m-gons into a convex container of minimum area with edges in n fixed directions. When the m-gons can be non-convex, the running time is $O((m^2+n)^{2k-2}(n+\log m))$, and when they are convex, the running times are $O((m+n)^{2k}(n+\log m))$ or $O(m^{k-1}(n^{2k+1}+\log m))$. Milenkovich and Daniels [22] described different algorithms for the same problems. Milenkovich [21] also studied the same problem when arbitrary rotations are allowed and the container is either a strip with a fixed width, a homothet of a given convex polygon, or an arbitrary rectangle (as in our work). He gave $(1+\varepsilon)$-approximation algorithms (no explicit running times are given, but they are apparently also exponential).

Some algorithms have been described for computing the packing of two or three convex polygons that minimizes the perimeter or area of the convex hull or the bounding box [1, 5, 17, 23].

Alt [2] demonstrated how a ρ-approximation algorithm for strip packing (axis-parallel rectangles with translations) can be turned into a $(1+\varepsilon)\rho$-approximation algorithm for the offline version of AREA-TRANSLATION, for any constant $\varepsilon > 0$. The same technique works for AREAROTATION. The idea is to apply the strip packing algorithm to strips of increasing widths and in the end choose the packing that resulted in the smallest area. Therefore, the same technique cannot be applied in the online setting, where we need to choose a placement for each piece and stick with it. Alt also mentioned that finding a minimum area bounding box of a set of convex polygons with arbitrary rotations allowed can be reduced to the problem where the pieces are axis-parallel rectangles with only translations allowed. This reduction increases the approximation ratio by a factor by 2. The reduction does not work when the pieces can be only translated, but Alt, de Berg, and Knauer [4] gave a 17.45-approximation algorithm for this problem using different techniques.

Lubachevsky and Graham [18] used computational experiments to find the rectangles of minimum area into which a given number $n \leq 5000$ of congruent circles can be packed; see also the follow-up work by Specht [24]. In another paper, Lubachevsky and Graham [19] studied the problem of minimizing the perimeter instead of the area.

Another fundamental packing problem is to find the smallest square containing a given number of *unit* squares, with arbitrary rotations allowed. A long line of mathematical research has been devoted to this problem, initiated by Erdős and Graham [12] in 1975, and it is still an active research area [10].

2 The perimeter versions

In Section 2.1, we present two online algorithms to minimize the perimeter of the bounding box: the algorithm BRICKTRANSLATION solves the problem PERIMETERTRANSLATION, where we can only translate pieces; the algorithm BRICKROTATION solves the problem PERIMETERROTATION, where also rotations are allowed. Both algorithms achieve a competitive ratio of 4. In Section 2.3, we show a lower bound of 4/3 for the version with translations and 5/4 for the version with rotations.

2.1 Algorithms to minimize perimeter

Algorithm for translations We pack the pieces into non-overlapping *bricks*; a technique first described by Januszewski and Lassak [15] which was also used by Fekete and Hoffmann [13] for the problem SQUAREINSQUAREAREA. Let a *k-brick* be a rectangle of size $\sqrt{2}^{-k} \times \sqrt{2}^{-k-1}$ if k is even and $\sqrt{2}^{-k-1} \times \sqrt{2}^{-k}$ if k is odd. A *brick* is a k-brick for some integer k.

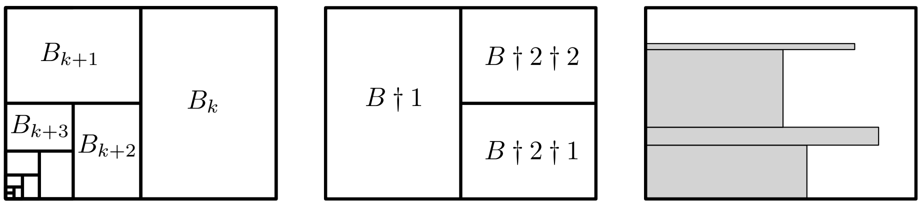


Figure 1: Left: Fundamental bricks. Middle: Splitting a brick. Right: Rectangular pieces packed in a brick.

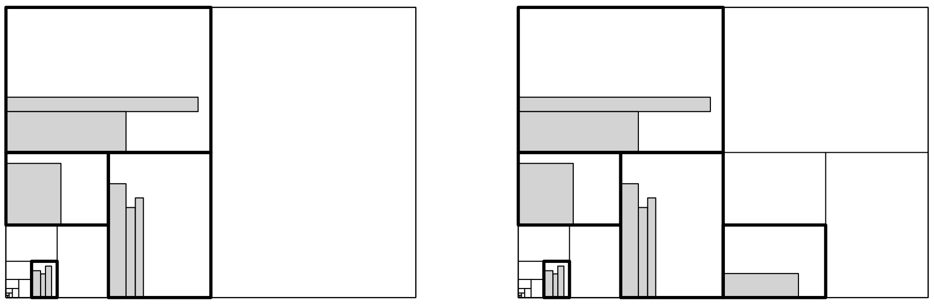


Figure 2: Left: Some pieces have been packed by the algorithm. The bricks in $\mathcal{D}$ are drawn with fat edges. Right: A new piece arrives. There is already a brick of the suitable size in $\mathcal{D}$, but there is not enough room, so a new brick of the same size is added to $\mathcal{D}$ where the piece is placed.

We tile the positive quadrant using one k-brick B_k for each integer k as in Figure 1 (left): if k is even, B_k is the k-brick with lower left corner $(0, \sqrt{2}^{-k-1})$ and otherwise, B_k is the k-brick with lower left corner $(\sqrt{2}^{-k-1}, 0)$. The bricks B_k are called the *fundamental* bricks. We define $B_{>k} := \bigcup_{i>k} B_i$ and $B_{\geq k} := B_{>k-1}$, so that $B_{>k}$ is the k-brick immediately below (if k is even) or to the left (if k is odd) of B_k.

An important property of a k-brick B is that it can be split into two $(k+1)$-bricks: $B \dagger 1$ and $B \dagger 2$; see Figure 1 (middle). We introduce a uniform naming and define $B \dagger 1$ to be the left half of B if k is even and the lower half of B if k is odd.

We define a *derived* brick recursively as follows: a derived brick is either (i) a fundamental brick B_k or (ii) $B \dagger 1$ or $B \dagger 2$, where B is a derived brick. We introduce an ordering $\prec$ of the derived k-bricks as follows. Consider two derived k-bricks D_1 and D_2 such that $D_1 \subset B_i$ and $D_2 \subset B_j$. If $i > j$, then $D_1 \prec D_2$. Else, if $i = j$ then the bricks D_1 and D_2 are both obtained by splitting the fundamental brick B_i, and the number of splits is $\ell := i - k$. Hence the bricks have the forms $D_1 = B_i \dagger b_{11} \dagger b_{12} \dagger \ldots \dagger b_{1\ell}$ and $D_2 = B_i \dagger b_{21}\, b_{22} \, \ldots \, b_{2\ell}$, where $b_{ij} \in \{1, 2\}$ for $i \in \{1, 2\}$ and $j \in \{1, \ldots, \ell\}$. We then define $D_1 \prec D_2$ if $(b_{11}, b_{12}, \ldots, b_{1\ell})$ precedes $(b_{21}, b_{22}, \ldots, b_{2\ell})$ in the lexicographic ordering.

We say that a k-brick is *suitable* for a piece p of size $w \times h$ if the width and height of the brick are at least w and h, respectively, and if that is not the case for a $(k+1)$-brick. We will always pack a given piece p in a derived k-brick that is suitable for p.

We now explain how we pack pieces into one specific brick; see Figure 1 (right). The first piece p that is packed in a brick B is placed with the lower left corner of p at the lower left corner of B. Suppose now that some other pieces $p_1, \ldots, p_i$ have been packed in B. If k is even, then $p_1, \ldots, p_i$ form a stack with the left edges contained in the left edge of B, and we place p on top of p_i (again, with the left edge of p contained in the left edge of B). Otherwise, $p_1, \ldots, p_i$ form a stack with the bottom edges contained in the bottom edge of B, and we place p to the right of p_i (again, with the bottom edge of p contained in the bottom edge of B). We say that a brick *has room* for a piece p if the packing scheme above places p within B, and it is apparent that an empty suitable brick for p has room for p.

The algorithm BrickTranslation maintains the collection $\mathcal{D}$ of non-overlapping derived bricks, such that one or more pieces have been placed in each brick in $\mathcal{D}$; see Figure 2. Before the first piece arrives, we set $\mathcal{D} := \emptyset$. Suppose that some stream of pieces have been packed, and that a new piece p appears. Choose k such that a k-brick is suitable for p. If there exists a derived k-brick $D \in \mathcal{D}$ such that

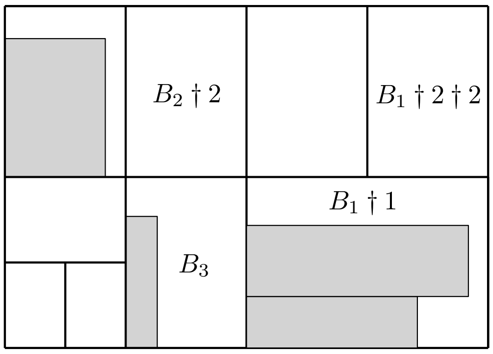


Figure 3: Brick B_3 is *sparse*, brick $B_2 \dagger 2$ is *empty*, brick $B_1 \dagger 1$ is *dense*. Brick $B_1 \dagger 2 \dagger 2$ is free, but not empty, since it is contained in $B_1 \dagger 2$.

D has room for p, then we pack p in D. Else, let D be the minimum derived k-brick (with respect to the ordering $\prec$ described before) such that D is interior-disjoint from each brick in $\mathcal{D}$; we then add D to $\mathcal{D}$ and pack p in D.

Theorem 2. *The algorithm* BrickTranslation *has a competitive ratio strictly less than 4 for* PerimeterTranslation.

Proof. We can assume, without loss of generality, that after we have packed the last rectangle, we have $\bigcup \mathcal{D} \subseteq B_{\geq 0}$ and $\bigcup \mathcal{D} \nsubseteq B_{\geq 1}$. As shown in Figure 3, we define a derived k-brick $B \subseteq B_{\geq 0}$ to be

- *sparse* if $B \in \mathcal{D}$ and the total height (if k is even, else width) of pieces stacked in B is less than half of the height (if k is even, else width) of B,
- *dense* if $B \in \mathcal{D}$ and B it is not sparse,
- *free* if B is interior-disjoint from each brick in $\mathcal{D}$, and
- *empty* if B is a maximal (w.r.t. inclusion) free brick.

Remark 3. Sparse, dense and empty bricks together cover $B_{\geq 0}$, in fact every brick in $\mathcal{D}$ is either sparse or dense, and any brick in $B_{\geq 0}$ that is interior-disjoint from bricks in $\mathcal{D}$ is contained in some empty brick.

Remark 4. Every k-brick $D \in \mathcal{D}$ contains pieces for which it is suitable. Therefore, if k is odd D contains a piece of height at least $\sqrt{2}^{-k}/2$, and if k is even D contains a piece of width at least $\sqrt{2}^{-k}/2$.

Remark 5. Every k-brick $D \in \mathcal{D}$ that is dense contains pieces with total area at least $1/4$ of the area of D. To see this, suppose that k is even, so that D is $\sqrt{2}^{-k} \times \sqrt{2}^{-k-1}$, then thanks to density the total height of pieces in D is at least half of its height, moreover thanks to Remark 4 all the pieces contained in D have width at least $\sqrt{2}^{-k}/2$. If k is odd we prove it analogously.

Remark 6. Consider two k-bricks M and N. If $M \prec N$ and M is free, then N is free. To prove this it is sufficient to consider the step in which the first piece p is placed within N and $N \dagger b_1 \dagger \ldots \dagger b_\ell$ is added to $\mathcal{D}$. Then, $N \dagger b_1 \dagger \ldots \dagger b_\ell$ should be the $\prec$-minimum free suitable k-brick, but $M \dagger b_1 \dagger \ldots \dagger b_\ell \prec N \dagger b_1 \dagger \ldots \dagger b_\ell$ gives a contradiction. It follows that whenever we have a set S of k-bricks that contains a free k-brick, then also $\max_\prec S$ is free. This turns out to be useful multiple times along the proof, choosing S to be the set of k-bricks not contained in a strictly larger empty brick.

Remark 7. There exists no empty 0-brick, otherwise $D \subseteq B_{\geq 1}$. Moreover, for every $k \geq 1$ we can have at most one sparse k-brick and one empty k-brick. In fact, a new empty (resp. sparse) k-brick is created only when no empty (resp. sparse) k-brick exists.

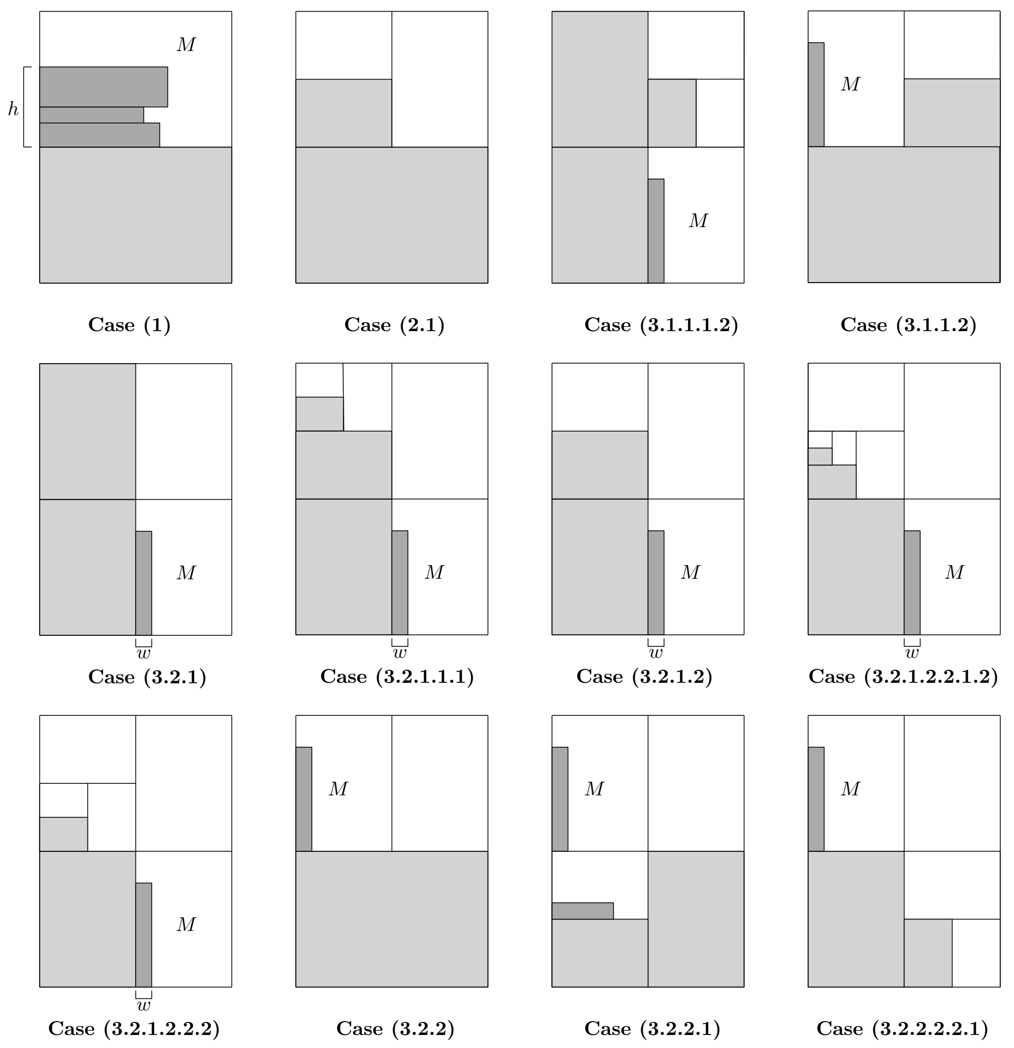


Figure 4: Some of the cases listed in the proof of Theorem 2 are shown. The grey area must fit within the bounding box considered in the case analysis.

In the following we prove an upper bound on the competitive ratio Alg/Opt, where Alg is the perimeter of the bounding box achieved by our online algorithm and Opt is the optimal perimeter computed offline. Hence, we need some techniques to provide an upper bound on Alg and a lower bound on Opt. For Alg, we will simply show a bounding box, in fact the perimeter of any bounding box containing all the pieces provides an upper bound to the minimum perimeter bounding box. For Opt, let A be the total area of pieces and L be the maximum length of an edge of a piece. If $L^2 > A$ then the minimum perimeter bounding box cannot have a smaller perimeter than a box of size $L \times A/L$. Otherwise, if $L^2 \leq A$ we have a weaker lower bound given by the box $\sqrt{A} \times \sqrt{A}$. Throughout the analysis we consider semiperimeters instead of perimeters to improve readability.

We denote with $A(\mathit{empty}), A(\mathit{sparse}), A(\mathit{dense})$ the total area of empty, sparse and dense bricks respectively. Thanks to Remark 3, we have that $A(\mathit{empty}) + A(\mathit{sparse}) + A(\mathit{dense}) = A(B_{\leq 0}) = \sqrt{2}$. We denote with A_{pcs} the total area of pieces in the stream. Thanks to Remark 5, we have $A_{pcs} \geq A(\mathit{dense})/4$. From now on the proof branches in many cases and subcases. We will perform a depth-first visit of the case tree, and for each leaf of this tree we will prove that the competitive ratio is strictly less than 4. Let k be the smallest integer such that there exists a k-brick in $\mathcal{D}$, and let $M \in \mathcal{D}$ be the $\prec$-maximal k-brick. From our assumptions, it follows that $k \geq 0$.

Case Tree **Case (1)** [*M is a 0-brick*]

Thanks to Remark 4 every piece in M has width at least 1/2. Let h be the total height of pieces stacked in M, then a bounding box of size size $1 \times (\sqrt{2}/2 + h)$ is obtained cutting the topmost part of M; see Figure 4. We can easily bound Opt with $1/2 \times h$, and we get

$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{1 + \frac{\sqrt{2}}{2} + h}{\frac{1}{2} + h} \leq 2 + \sqrt{2} < 4.$$

Case (2) [*M is a k-brick for $k \geq 2$*]

Here we have two cases.

Case (2.1) [*There exist a 1-brick N_1 and a 2-brick N_2 that are empty*]

Thanks to Remark 6 we can choose $N_1 = B_0 \dagger 2$ and $N_2 = B_0 \dagger 1 \dagger 2$. In fact, for $B_0 \dagger 2$ it is sufficient to choose S as the set of all 1-bricks, while for $B_0 \dagger 1 \dagger 2$ we can choose S to be the set of all 2-bricks that are not contained in a larger free brick. Thus, we can cut the topmost half of B_0 and get $\textsc{Alg} \leq 1 + 3/4 \cdot \sqrt{2}$; see Figure 4. We have

$$A(\textit{empty}) \leq \sum_{i \geq 1} A(B_i) \leq \frac{\sqrt{2}}{2}$$

$$A(\textit{sparse}) \leq \sum_{i \geq 2} A(B_i) = \frac{\sqrt{2}}{4} \quad \text{(thanks to case (2) clause there is no sparse 1-brick)}$$

$$A_{pcs} \geq \frac{A(\textit{dense})}{4} \geq \frac{A(B_{\geq 0}) - A(\textit{sparse}) - A(\textit{empty})}{4} \geq \frac{\sqrt{2}}{16}$$

Now we are ready to bound Opt:

$$\textsc{Opt} \geq 2 \cdot \sqrt{A_{pcs}} = \sqrt{\frac{\sqrt{2}}{4}}$$

$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{1 + \frac{3}{4}\sqrt{2}}{\sqrt{\frac{\sqrt{2}}{4}}} \approx 3.47 < 4.$$

Case (2.2) [*For $j = 1$ or $j = 2$ there does not exist an empty j-brick*]

In this case we just use $\textsc{Alg} \leq 1 + \sqrt{2}$. Then we have

$$A(\textit{empty}) \leq \sum_{i \geq 1 \wedge i \neq j} A(B_i) \leq \frac{3}{8}\sqrt{2} \quad \text{(worst case is when } j = 2\text{)}$$

$$A(\textit{sparse}) \leq \sum_{i \geq 2} A(B_i) = \frac{\sqrt{2}}{4}$$

therefore performing the same computations of case (2.1), $A_{pcs} \geq 3/32 \cdot \sqrt{2}$, and finally

$$\textsc{Opt} \geq 2 \cdot \sqrt{\frac{3}{32}\sqrt{2}} = \sqrt{\frac{3}{8}\sqrt{2}}$$

$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{1 + \sqrt{2}}{\sqrt{\frac{3}{8}\sqrt{2}}} \approx 3.32 < 4.$$

Case (3) [*M is a 1-brick*]

For the rest of the proof L will be the length of the longest edge among all pieces. Since M is a 1-brick, we have $\sqrt{2}/4 < L \leq \sqrt{2}/2$. Here we have two cases.

Case (3.1) [*There does not exists an empty 1-brick*]

Here we have two cases.

Case (3.1.1) [*For $j = 2$ and $j = 3$ there exists an empty j-brick*]

Here we have three cases.

Case (3.1.1.1) [*M is the fundamental brick B_1*]

Thanks to Remark 6 we can assume $B_0 \dagger 2 \dagger 2$ and $B_0 \dagger 2 \dagger 1 \dagger 2$ to be empty. Here we have two cases.
Case (3.1.1.1.1) [*M is dense*]
Since $M = B_1$ is the $\prec$-maximal k-brick in $\mathcal{D}$, then there does not exist a sparse 1-brick.

$$A(\mathit{empty}) \leq \sum_{i\geq 2} A(B_i) \leq \frac{\sqrt{2}}{4}$$
$$A(\mathit{sparse}) \leq \sum_{i\geq 2} A(B_i) = \frac{\sqrt{2}}{4}$$

therefore $A_{pcs} \geq \frac{\sqrt{2}}{8}$, and finally

$$\textsc{Opt} \geq 2 \cdot \sqrt{\frac{\sqrt{2}}{8}} = \sqrt{\frac{\sqrt{2}}{2}}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{1+\sqrt{2}}{\sqrt{\frac{\sqrt{2}}{2}}} \approx 2.87 < 4.$$

Case (3.1.1.1.2) [*M is sparse*]
Then, we can cut the rightmost part of $B_{\geq 0}$ and get a $3/4 \times \sqrt{2}$ bounding box; see Figure 4. We have

$$A(\mathit{empty}) \leq \sum_{i\geq 2} A(B_i) \leq \frac{\sqrt{2}}{4}$$
$$A(\mathit{sparse}) \leq \sum_{i\geq 1} A(B_i) \leq \frac{\sqrt{2}}{2}$$

hence $A_{pcs} \geq \sqrt{2}/16$. Since $L^2 > 1/8 > \sqrt{2}/16$ we finally have

$$\textsc{Opt} \geq L + \frac{A_{pcs}}{L} \geq \frac{\sqrt{2}}{4} + \frac{1}{4} \quad \text{(minimizing over } L \in [\sqrt{2}/4, \sqrt{2}/2])$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{3/4+\sqrt{2}}{\frac{\sqrt{2}}{4}+\frac{1}{4}} \approx 3.59 < 4.$$

Case (3.1.1.2) [$M = B_0 \dagger 1$]
Thanks to Remark 6 we can assume $B_0 \dagger 2 \dagger 2$ to be empty. Then, we can cut the topmost part of $B_{\geq 0}$ and get a $1 \times \sqrt{2}/2 + L$ bounding box; see Figure 4. We have

$$A(\mathit{empty}) \leq \sum_{i\geq 2} A(B_i) \leq \frac{\sqrt{2}}{4}$$
$$A(\mathit{sparse}) \leq \sum_{i\geq 1} A(B_i) \leq \frac{\sqrt{2}}{2}$$

hence $A_{pcs} \geq \sqrt{2}/16$. Since $L^2 > 1/8 > \sqrt{2}/16$ we finally have

$$\textsc{Opt} \geq L + \frac{A_{pcs}}{L} \geq L + \frac{\sqrt{2}}{16L}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{1+\sqrt{2}/2+L}{L+\frac{\sqrt{2}}{16L}} \leq 2+\sqrt{2} < 4. \quad \text{(maximizing over } L \in [\sqrt{2}/4, \sqrt{2}/2])$$

Case (3.1.1.3) [$M = B_0 \dagger 2$]
This case is analogous to the previous one, in fact thanks to Remark 6 we can assume $B_0 \dagger 1 \dagger 2$ to be empty and cut the topmost part of $B_{\geq 0}$.
Case (3.1.2) [*For $j = 2$ or $j = 3$ there does not exist an empty j-brick*]

$$A(\mathit{empty}) \leq \sum_{i\geq 2 \wedge i\neq j} A(B_i) \leq \frac{3}{16}\sqrt{2} \quad \text{(worst case is when } j=3\text{)}$$
$$A(\mathit{sparse}) \leq \sum_{i\geq 1} A(B_i) \leq \frac{\sqrt{2}}{2}$$

hence $A_{pcs} \geq \frac{5}{64}\cdot\sqrt{2}$. Since $L^2 > 1/8 > \frac{5}{64}\cdot\sqrt{2}$ we finally have

$$\textsc{Opt} \geq L + \frac{A_{pcs}}{L} \geq \frac{\sqrt{2}}{4} + \frac{5}{16} \quad \text{(minimizing over } L \in [\sqrt{2}/4, \sqrt{2}/2]\text{)}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{1+\sqrt{2}}{\frac{\sqrt{2}}{4}+\frac{5}{16}} \approx 3.62 < 4.$$

Case (3.2) [*There exists an empty* 1*-brick*]

Thanks to Remark 6 we can assume $B_0 \dagger 2$ to be empty. Here we have two cases.

Case (3.2.1) [*M is the fundamental brick B_1*]

Let w be the total width of pieces stacked in M. Since $B_0 \dagger 2$ is empty, we can cut the rightmost part of $B_{\geq 0}$ and get a $(1/2+w)\times\sqrt{2}$ bounding box; see Figure 4. Since increasing w only improves our estimates, we consider the corner case $w=0$. Now we have two cases.

Case (3.2.1.1) [*There does not exist an empty* 2*-brick*]

Here we have two cases.

Case (3.2.1.1.1) [*For $j=3$ and $j=4$ there exists an empty j-brick*]

Thanks to Remark 6 we can assume $B_0 \dagger 1 \dagger 2 \dagger 2$ and $B_0 \dagger 1 \dagger 2 \dagger 1 \dagger 2$ to be empty. Thus, we can cut the topmost part of $B_{\geq 0}$ and get a $1/2 \times (7/8\cdot\sqrt{2})$ bounding box; see Figure 4. We have

$$A(\mathit{empty}) \leq \sum_{i\geq 1 \wedge i\neq 2} A(B_i) \leq \frac{3}{8}\sqrt{2}$$
$$A(\mathit{sparse}) \leq \sum_{i\geq 1} A(B_i) \leq \frac{\sqrt{2}}{2}$$

hence $A_{pcs} \geq \sqrt{2}/32$. Since $L^2 > 1/8 > \sqrt{2}/32$ we finally have

$$\textsc{Opt} \geq L + \frac{A_{pcs}}{L} \geq \frac{\sqrt{2}}{4} + \frac{1}{8}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{1/2 + (7/8\cdot\sqrt{2})}{\frac{\sqrt{2}}{4}+\frac{1}{8}} \approx 3.63 < 4.$$

Case (3.2.1.1.2) [*For $j=3$ or $j=4$ there does not exist an empty j-brick*]

$$A(\mathit{empty}) \leq \sum_{i\geq 1 \wedge i\neq 2,j} A(B_i) \leq \frac{11}{32}\sqrt{2} \quad \text{(worst case is when } j=4\text{)}$$
$$A(\mathit{sparse}) \leq \sum_{i\geq 1} A(B_i) \leq \frac{\sqrt{2}}{2}$$

hence $A_{pcs} \geq 5/128\cdot\sqrt{2}$. Since $L^2 > 1/8 > 5/128\cdot\sqrt{2}$ we finally have

$$\textsc{Opt} \geq L + \frac{A_{pcs}}{L} \geq \frac{\sqrt{2}}{4} + \frac{5}{32}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{1/2+\sqrt{2}}{\frac{\sqrt{2}}{4}+\frac{5}{32}} \approx 3.75 < 4.$$

Case (3.2.1.2) [*There exists an empty* 2*-brick*]

Thanks to Remark 6 we can assume $B_0 \dagger 1 \dagger 2$ to be empty. Thus, we can cut the topmost part of $B_{\geq 0}$ and get a $1/2 \times (3/4 \cdot \sqrt{2})$ bounding box; see Figure 4. Here we have two cases.

Case (3.2.1.2.1) [*There does not exist an empty* 3*-brick*]

$$A(empty) \leq \sum_{i \geq 1 \wedge i \neq 3} A(B_i) \leq \frac{7}{16}\sqrt{2}$$
$$A(sparse) \leq \sum_{i \geq 1} A(B_i) \leq \frac{\sqrt{2}}{2}$$

hence $A_{pcs} \geq \sqrt{2}/64$. Since $L^2 > 1/8 > \sqrt{2}/64$ we finally have

$$\textsc{Opt} \geq L + \frac{A_{pcs}}{L} \geq \frac{\sqrt{2}}{4} + \frac{1}{16}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{\frac{1}{2} + \frac{3}{4}\sqrt{2}}{\frac{\sqrt{2}}{4} + \frac{1}{16}} \approx 3.75 < 4.$$

Case (3.2.1.2.2) [*There exists an empty* 3*-brick*]

Thanks to Remark 6 we can assume $B_0 \dagger 1 \dagger 1 \dagger 2$ to be empty. Here we have two cases.

Case (3.2.1.2.2.1) [*There does not exist an empty* 4*-brick*]

Here we have two cases.

Case (3.2.1.2.2.1.1) [*For $j = 5$ or $j = 6$ there does not exist an empty j-brick*]

$$A(empty) \leq \sum_{i \geq 1 \wedge i \neq 4, j} A(B_i) \leq \frac{59}{128}\sqrt{2} \quad \text{(worst case is when } j = 6)$$
$$A(sparse) \leq \sum_{i \geq 1} A(B_i) \leq \frac{\sqrt{2}}{2}$$

hence $A_{pcs} \geq 5/512 \cdot \sqrt{2}$. Since $L^2 > 1/8 > 5/512 \cdot \sqrt{2}$ we finally have

$$\textsc{Opt} \geq L + \frac{A_{pcs}}{L} \geq \frac{\sqrt{2}}{4} + \frac{5}{128}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{\frac{1}{2} + \frac{3}{4}\sqrt{2}}{\frac{\sqrt{2}}{4} + \frac{5}{128}} \approx 3.98 < 4.$$

Case (3.2.1.2.2.1.2) [*For $j = 5$ and $j = 6$ there exists an empty j-brick*]

Thanks to Remark 6 we can assume $B_0 \dagger 1 \dagger 1 \dagger 1 \dagger 2 \dagger 2$ and $B_0 \dagger 1 \dagger 1 \dagger 1 \dagger 2 \dagger 1 \dagger 2$ to be empty. Then, we can cut the topmost part of $B_{\geq 0}$ and get a $1/2 \times (11/16 \cdot \sqrt{2})$ bounding box; see Figure 4. We have

$$A(empty) \leq \sum_{i \geq 1 \wedge i \neq 4} A(B_i) \leq \frac{15}{32}\sqrt{2}$$
$$A(sparse) \leq \sum_{i \geq 1} A(B_i) \leq \frac{\sqrt{2}}{2}$$

hence $A_{pcs} \geq \sqrt{2}/128$. Since $L^2 > 1/8 > \sqrt{2}/128$ we finally have

$$\textsc{Opt} \geq L + \frac{A_{pcs}}{L} \geq \frac{\sqrt{2}}{4} + \frac{1}{32}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{\frac{1}{2} + \frac{11}{16}\sqrt{2}}{\frac{\sqrt{2}}{4} + \frac{1}{32}} \approx 3.83 < 4.$$

Case (3.2.1.2.2.2) [*There exists an empty* 4*-brick*]

Thanks to Remark 6 we can assume $B_0 \dagger 1 \dagger 1 \dagger 1 \dagger 2$ to be empty. Then, we can cut the topmost part of $B_{\geq 0}$ and get a $1/2 \times (5/8 \cdot \sqrt{2})$ bounding box; see Figure 4. Now it remains to bound Opt, and we just assume $\textsc{Opt} \geq L \geq \sqrt{2}/4$, finally

$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{\frac{1}{2} + \frac{5}{8}\sqrt{2}}{\frac{\sqrt{2}}{4}} \approx 3.92 < 4.$$

Case (3.2.2) $[M = B_0 \dagger 1]$

For the rest of the proof let L be the length of the longest of pieces' edges then, according to Remark 4, $\sqrt{2}/4 \leq L \leq \sqrt{2}/2$. We can cut the topmost part of $B_{\geq 0}$ and get a $1 \times (\sqrt{2}/2 + L)$ bounding box; see Figure 4. Here we have two cases.

Case (3.2.2.1) [*There exists a* 2*-brick in* $\mathcal{D}$]

Thanks to Remark 4, we have a piece of width at least 1/4, and combining this with the fact that we have a piece of height L, it is apparent that $\textsc{Opt} \geq 1/4 + L$; see Figure 4. Thus,

$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{1 + \frac{\sqrt{2}}{2} + L}{\frac{1}{4} + L} \leq 2 + \sqrt{2} < 4 \quad \text{(maximizing over } L \in [\sqrt{2}/4, \sqrt{2}/2]).$$

Case (3.2.2.2) [*There does not exist a* 2*-brick in* $\mathcal{D}$]

Here we have two cases.

Case (3.2.2.2.1) [*There does not exist an empty* 2*-brick*]

$$A(\mathit{empty}) \leq \sum_{i \geq 1 \wedge i \neq 2} A(B_i) \leq \frac{3}{8}\sqrt{2}$$
$$A(\mathit{sparse}) \leq \sum_{i \geq 1} A(B_i) \leq \frac{3}{8}\sqrt{2}$$

hence $A_{pcs} \geq \sqrt{2}/16$. Since $L^2 > 1/8 > \sqrt{2}/16$ we finally have

$$\textsc{Opt} \geq L + \frac{A_{pcs}}{L} \geq L + \frac{\sqrt{2}}{16L}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{1 + \frac{\sqrt{2}}{2} + L}{L + \frac{\sqrt{2}}{16L}} \leq 2 + \sqrt{2} < 4 \quad \text{(maximizing over } L \in [\sqrt{2}/4, \sqrt{2}/2]).$$

Case (3.2.2.2.2) [*There exists an empty* 2*-brick*] Thanks to Remark 6 we can assume $B_1 \dagger 2$ to be empty. Here we have two cases.

Case (3.2.2.2.2.1) [*There exists an empty* 3*-brick*]

Thanks to Remark 6 we can assume $B_1 \dagger 1 \dagger 2$ to be empty. Then, we can cut the rightmost part of $B_{\geq 0}$ and get a $3/4 \times (\sqrt{2}/2 + L)$ bounding box; see Figure 4. Now it remains to bound Opt. We have

$$A(\mathit{empty}) \leq \sum_{i \geq 1} A(B_i) \leq \frac{\sqrt{2}}{2}$$
$$A(\mathit{sparse}) \leq \sum_{i \geq 1 \wedge i \neq 2} A(B_i) \leq \frac{3}{8}\sqrt{2}$$

hence $A_{pcs} \geq \sqrt{2}/32$. Since $L^2 > 1/8 > \sqrt{2}/32$ we finally have

$$\textsc{Opt} \geq L + \frac{A_{pcs}}{L} \geq L + \frac{\sqrt{2}}{32L}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{\frac{3}{4} + \frac{\sqrt{2}}{2} + L}{L + \frac{\sqrt{2}}{32L}} \leq 3.79 < 4 \quad \text{(maximizing over } L \in [\sqrt{2}/4, \sqrt{2}/2]).$$

Case (3.2.2.2.2.2) [*There does not exist an empty* 3*-brick*]

$$A(empty) \le \sum_{i\ge 1 \wedge i\neq 3} A(B_i) \le \frac{7}{16}\sqrt{2}$$
$$A(sparse) \le \sum_{i\ge 2 \wedge i\neq 2} A(B_i) \le \frac{3}{8}\sqrt{2}$$

hence $A_{pcs} \ge 3/64 \cdot \sqrt{2}$. Since $L^2 > 1/8 > 3/64 \cdot \sqrt{2}$ we finally have

$$\textsc{Opt} \ge L + \frac{A_{pcs}}{L} \ge L + \frac{3\sqrt{2}}{64L}$$
$$\frac{\textsc{Alg}}{\textsc{Opt}} \le \frac{1+\frac{\sqrt{2}}{2}+L}{L+\frac{3\sqrt{2}}{64L}} \le 3.82 < 4 \quad \text{(maximizing over } L \in [\sqrt{2}/4, \sqrt{2}/2]).$$

□

Algorithm using rotations The algorithm BrickRotation is almost identical to BrickTranslation, but with the difference that we rotate each piece so that its height is at least its width.

Theorem 8. *The algorithm* BrickRotation *has a competitive ratio of strictly less than 4 for* PerimeterRotation.

Proof. The analysis of BrickTranslation carried out in the proof of Theorem 2 still holds, in fact all the estimates on Opt derived from consideration about area are still valid, and the only delicate spot is case (3.2.2.1). In that case we assume to have a piece p having an edge of length $L \in [\sqrt{2}/4, \sqrt{2}/2]$, and that there exists a 2-brick in $\mathcal{D}$. Thanks to Remark 4 there exists a piece q of size $w_q \times h_q$ with $w_q \ge 1/4$, moreover we rotate every piece so that $1/4 \le w_q \le h_q$. Finally, a box that contains both p and q must have size at least $L \times w_q$ or $L \times h_q$, hence

$$\textsc{Opt} \ge \min\{L + w_q, L + h_q\} \ge L + \frac{1}{4}.$$

This gives exactly the same bound showed in case (3.2.2.1) and completes the proof. □

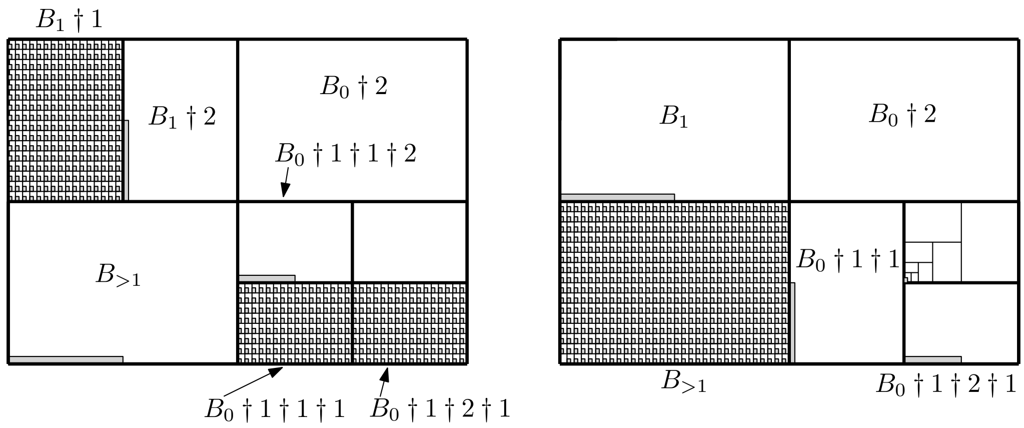
2.2 A similar but inferior algorithm

Here we consider the algorithm we get by making a slight change to BrickTranslation. Suppose that the very first piece p arrives and that a k-brick is suitable for p. Instead of placing p in B_k (as BrickTranslation would do), we consider the brick $B_{>k}$ to be a fundamental brick (although in the original algorithm, it was an infinite union of fundamental bricks) and we place p in $B_{>k}$. Thus, we are never going to use the fundamental bricks B_i individually, for $i > k$. From here on, the algorithm does as BrickTranslation: Whenever a new piece arrives, we place it in the first derived brick of the suitable size that has room. This behavior is similar to the algorithm for the problem SquareInSquareArea that was described by Fekete and Hoffmann [13]. That problem is studied in more detail in Section 3.3, and for that problem, the algorithm seems to be no worse than ours.

Interestingly, the following theorem together with Theorem 2 implies that the modified algorithm is worse for the problem PerimeterTranslation.

Theorem 9. *The modified version of* BrickTranslation *has a competitive ratio of at least* 4 *for the problem* PerimeterTranslation.

Proof. For any $\varepsilon' > 0$, we can make an instance realizing a competitive ratio of more than $4 - \varepsilon'$ as follows. Figure 5 shows the packing produced by the modified and the original algorithm. We first give the algorithm the rectangle $(1/2\sqrt{2} + \varepsilon) \times \varepsilon$ for an infinitesimal $\varepsilon > 0$. The rectangle is placed in $B_{>1}$ by the modified algorithm. For a large odd integer k, we then feed the algorithm with small rectangles of size $(\sqrt{2}^{-k-1} + \varepsilon) \times (\sqrt{2}^{-k} + \varepsilon)$ until $B_1 \dagger 1$ has been completely split into $(k-2)$-bricks, each of which contains one small rectangle. We now give the algorithm a piece of size $\varepsilon \times (1/4 + \varepsilon)$, which is



$B_1 \dagger 1$
$B_1 \dagger 2$
$B_0 \dagger 2$
$B_0 \dagger 1 \dagger 1 \dagger 2$
$B_{>1}$
$B_0 \dagger 1 \dagger 1 \dagger 1$
$B_0 \dagger 1 \dagger 2 \dagger 1$
B_1
$B_0 \dagger 2$
$B_0 \dagger 1 \dagger 1$
$B_{>1}$
$B_0 \dagger 1 \dagger 2 \dagger 1$

Figure 5: Left: A configuration produced by the modified version of BrickTranslation. Right: The configuration produced by the original algorithm BrickTranslation.

placed in $B_1 \dagger 2$. We again give the algorithm many small rectangles until $B_0 \dagger 1 \dagger 1 \dagger 1$ has been split into $(k-2)$-bricks. Now follows a rectangle of size $(1/4\sqrt{2}+\varepsilon) \times \varepsilon$, which is placed in $B_0 \dagger 1 \dagger 1 \dagger 2$. Finally, we fill $B_0 \dagger 1 \dagger 2 \dagger 1$ with small rectangles.

Note that as $k \longrightarrow \infty$, the bounding box of the produced packing converges to $B_{\geq 0}$, so it has a perimeter as B_{-1}. On the other hand, observe that as $\varepsilon \longrightarrow 0$, we have $\Sigma \longrightarrow A(B_1)/4 = A(B_3)$, since the small rectangles fill out bricks with a total area of $A(B_1)$ and with density 1/4. In the limit, all the pieces can actually be packed into B_3, so Opt is at most the perimeter of B_3. But the perimeter of B_{-1} is 4 times that of B_3, which finishes the proof. □

2.3 Lower bounds

Lemma 10. *Consider any algorithm A for the problem* PerimeterTranslation. *Then the competitive ratio of A is at least* 4/3.

Proof. We first feed A with two unit squares. Let the bounding box of the two squares have size $a \times b$ and suppose without loss of generality that $a \leq b$. Then $a \geq 1$ and $b \geq 2$. We now give A a rectangle of size $2 \times \varepsilon$ for a small value $\varepsilon > 0$. The produced packing has a bounding box of perimeter more than 8, whereas the optimal has perimeter $6 + 2\varepsilon$. Therefore, the competitive ratio is $\frac{8}{6+2\varepsilon} = \frac{4}{3+\varepsilon}$. By letting $\varepsilon \longrightarrow 0$, we get that the ratio is at least 4/3. □

Lemma 11. *Consider any algorithm A for the problem* PerimeterRotation. *Then the competitive ratio of A is at least* 5/4.

Proof. We first feed A with three unit squares. Let the bounding box of the three squares have size $a \times b$ and suppose without loss of generality that $a \leq b$. Suppose first that $b < 3$. Then we must have $a \geq 2$ for the box to contain the squares. We then give the algorithm the rectangle $\varepsilon \times 3$ for a small value $\varepsilon > 0$. The produced packing has a bounding box of size at least $(2+\varepsilon) \times 3$ and perimeter more than 10, while the optimal solution has size $(1+\varepsilon) \times 3$ and perimeter $8 + 2\varepsilon$.

On the other hand, if $b \geq 3$, we give the algorithm one more unit square. The produced packing has a bounding box of size at least 2×3 or at least 1×4, and thus perimeter at least 10, while the optimal packing has size 2×2 and perimeter 8.

We get that the competitive ratio is at least $\frac{10}{8+2\varepsilon} = \frac{5}{4+\varepsilon}$, and by letting $\varepsilon \longrightarrow 0$, we get that the ratio is at least 5/4. □

3 Area versions

3.1 General lower bounds

In this section we show that, if we allow pieces to be arbitrary rectangles, we cannot bound the competitive ratio for neither AreaTranslation nor AreaRotation as a function of the area Opt of the optimal packing. However we will be able to bound the competitive ratio as a function of the total number n of pieces in the stream.

Lemma 12. *Consider any algorithm A solving* AreaTranslation *or* AreaRotation *and let any $m \in \mathbb{N}$ and $p \in \mathbb{R}$ be given. There exists a stream of $n = m^2 + 1$ rectangles such that (i) the rectangles can be packed into a bounding box of area $2p^2$, and (ii) algorithm A produces a packing with a bounding box of area at least mp^2.*

Proof. We first feed A with m^2 rectangles of size $p \times \frac{p}{m^2}$. These rectangles have total area p^2. Let $a \times b$ be the size of the bounding box of the produced packing.

Suppose first that $a \geq \frac{p}{m}$ and $b \geq \frac{p}{m}$ hold. We then feed A with a long rectangle of size $pm^2 \times \frac{p}{m^2}$. The produced packing has a bounding box of area at least $\frac{p}{m} \cdot pm^2 = mp^2$. The optimal packing is to pack the m^2 small rectangles along the long rectangle, which would produce a packing with bounding box of size $pm^2 \times \frac{2p}{m^2} = 2p^2$.

Otherwise, we must have $b > pm$ or $a > pm$, since $ab \geq p^2$. We then feed A with a square of size $p \times p$. The produced packing has a bounding box of area at least $p \cdot pm = mp^2$. The optimal packing is obtained stacking the m^2 thin rectangles on top of the big square, which produces a packing with bounding box of size $p \times 2p = 2p^2$. □

Corollary 13. *Let A be an algorithm for* AreaTranslation *or* AreaRotation. *Then A does not have an asymptotic, and hence also absolute, competitive ratio which is a function of* Opt.

Proof. Let f be any function of Opt. For any value Opt $= c$, we choose $p := \sqrt{c/2}$. We now choose $m > 2f(c)$ and obtain that the competitive ratio is at least $\frac{mp^2}{2p^2} = m/2 > f(c) = f(\text{Opt})$. □

Corollary 14. *Let A be an algorithm for* AreaTranslation *or* AreaRotation. *If A has an asymptotic competitive ratio of $f(n)$, where $n = |L|$ is the number of pieces in the stream, then $f(n) = \Omega(\sqrt{n})$. This holds even when all edges of the pieces are required to have length at least* 1.

Proof. We choose $p := m^2$. Then all edges have length at least 1, and the competitive ratio is at least $\frac{mp^2}{2p^2} = m/2 = \Omega(\sqrt{n})$. Here, Opt can be arbitrarily big by choosing m big enough, so it is a lower bound on the asymptotic competitive ratio. □

3.2 Algorithms for arbitrary pieces

In this section we provide algorithms that solve AreaTranslation and AreaRotation with a competitive ratio of $O(\sqrt{n})$, where n is the total number of pieces. Thus we match the bounds provided in the previous section.

We first describe the algorithm DynBoxTrans that solves AreaTranslation. We assume to receive a stream of pieces $p_1, \ldots, p_n$ of unknown length n, such that piece p_i has size $w_i \times h_i$. For each $k \in \mathbb{Z}$, we define a rectangular box B_k with a size varying dynamically. After pieces $p_1, \ldots, p_j$ have been processed B_k has size $2^k \times T_j$, where $T_j := H_j\sqrt{j} + 7H_j$ and $H_j := \max_{i=1,\ldots,j} h_i$. We place the boxes with their bottom edges on the x-axis and in order such that the right edge of B_{k-1} is contained in the left edge of B_k; see Figure 6. Furthermore, we place the lower left corner of box B_0 at the point $(1, 0)$. It then holds that all the boxes are to the right of the point $(0, 0)$.

We say that the box B_k is *wide enough* for a piece $p_i = w_i \times h_i$ if $w_i \leq 2^k$. If a box B_k is wide enough for p_i, we can pack p_i in B_k using the online strip packing algorithm NFS$_k$ that packs rectangles into a strip of width 2^k. The algorithm NFS$_k$ is the *next-fit shelf algorithm* first described by Baker and Schwartz [6]. The algorithm packs pieces in *shelves* (rows), and each shelf is given a fixed height of 2^j for some $j \in \mathbb{Z}$ when it is created; see Figure 7. The width of each shelf is 2^k, since this is the width of the box B_k.

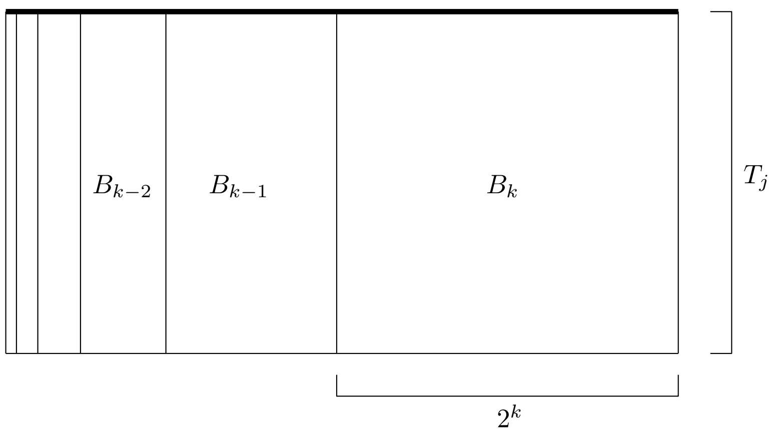


Figure 6: The algorithm DynBoxTrans packs pieces into the boxes B_k that form a row. Every box has height T_j that is dynamically updated.

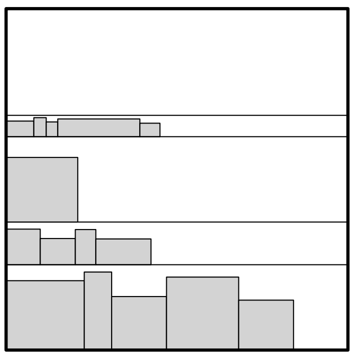


Figure 7: A packing produced by the next-fit shelf algorithm using four shelves.

A piece of height h, where $2^{j-1} < h \leq 2^j$, is packed in a shelf of height 2^j. We divide the shelves into two types. If the total width of pieces in a shelf is more than 2^{k-1} we call that shelf *dense*, otherwise we say it is *sparse*. The algorithm NFS_k places each piece as far left as possible into the currently sparse shelf of the proper height. If there is no sparse shelf of this height or the sparse shelf has not room for the piece, a new shelf of the appropriate height is created on top of the top shelf, and the piece is placed there at the left end of this new shelf. This ensures that at any point in time there exists at most one sparse shelf for each height 2^j.

If we allow the height of the box B_k to grow large enough with respect to shelves' heights, the space wasted by sparse shelves becomes negligible and we obtain a constant density strip packing, as stated in the following lemma.

Lemma 15. *Let $\widetilde{H}$ be the total height of shelves in B_k, and H_{max} be the maximum height among pieces in B_k. If $\widetilde{H} \geq 6H_{max}$, then the pieces in B_k are packed with density at least $1/12$.*

Proof. Let $2^{m-1} < H_{max} \leq 2^m$, so that $\widetilde{H} \geq 3 \cdot 2^m$. For each $i \leq m$ we have at most one sparse shelf of height 2^i and each shelf of B_k has height at most 2^m, hence the total height of sparse shelves is at most $\sum_{i \leq m} 2^i = 2^{m+1}$, so the total height of dense shelves is at least $\widetilde{H} - 2^{m+1} \geq \widetilde{H}/3$. Thus, the total area of the dense shelves is at least $2^k \cdot \widetilde{H}/3$.

Consider a dense shelf of height 2^i. Into that shelf, we have packed pieces of height at least 2^{i-1}, and the total width of these pieces is at least 2^{k-1}. Hence, the density of pieces in the shelf is at least $1/4$. Therefore, the total area of pieces in B_k is at least $2^k \cdot \widetilde{H}/12$. On the other hand, the area of the bounding box is $2^k \cdot \widetilde{H}$, that yields the desired density. □

Now we are ready to describe how the algorithm works. When the first piece p_1 arrives, let $2^{k-1} < w_1 \leq 2^k$, then we pack it in the box B_k according to NFS_k and define B_k to be the *active box*. Suppose now that B_i is the active box when the piece p_j arrives, first we update the value of the threshold T_{j-1} to T_j, then we have two cases. If $w_j > 2^i$ we choose ℓ such that $2^{\ell-1} < w_j \leq 2^\ell$, pack p_j in B_ℓ and define B_ℓ to be the active box. Else, B_i is wide enough for p_j and we try to pack p_j into B_i. Since B_i has size $2^i \times T_j$ it may happen that NFS_i exceeds the threshold T_j while packing p_j, generating an overflow. In this case, instead of packing p_j in B_i, we pack p_j into B_{i+1} and define that to be the active box.

Theorem 16. *The algorithm* DynBoxTrans *has an absolute competitive ratio of $O(\sqrt{n})$ for the problem* AreaTranslation *on a stream of n pieces.*

Proof. First, define Σ_j as the total area of the first j pieces, $W := \max_{i=1,\ldots,n} w_i$ and recall that $H_j = \max_{i=1,\ldots,j} h_i$ and $T_j = H_j\sqrt{n} + 7H_j$. Let B_k be the last active box, so that we can enclose all the pieces in a bounding box of size $2^{k+1} \times T_n$, and bound the area returned by the algorithm as $\textsc{Alg} = O(2^k H_n \sqrt{n})$. On the other hand we are able to bound the optimal offline packing as $\textsc{Opt} = \Omega(\Sigma_n + WH_n)$.

If the active box never changed, then we have $2^k < 2W$ that implies $\textsc{Alg} = O(WH_n\sqrt{n}) = \textsc{Opt} \cdot O(\sqrt{n})$. Otherwise, let B_ℓ be the last active box before B_k, and p_j be the first piece put in B_k. Here we have two cases.

Case (1) $[w_j > 2^\ell]$ In this case we have $2^k < 2W$ that implies $\textsc{Alg} = O(WH_n\sqrt{n}) = \textsc{Opt} \cdot O(\sqrt{n})$.

Case (2) $[w_j \leq 2^\ell]$ In this case we have $k = \ell + 1$. Denote with $\widetilde{H_i}$ the total height of shelves in B_i. Then we have $\widetilde{H_\ell} \geq T_j - H_j = H_j\sqrt{n} + 6H_j$, otherwise we could pack p_j in B_ℓ. Thus, we can apply Lemma 15 and conclude that the box B_ℓ of size $2^\ell \times T_j$ is filled with constant density. Here we have two cases.

Case (2.1) $[\widetilde{H_k} \leq T_j]$ In this case we have $\textsc{Alg} = O(2^k T_j)$ and, thanks to the constant density packing of B_ℓ we have $\Sigma_j = \Theta(2^\ell \widetilde{H_\ell}) = \Theta(2^k T_j)$. Since $\textsc{Opt} \geq \Sigma_j$, we get $\textsc{Alg} = O(\textsc{Opt})$.

Case (2.2) $[\widetilde{H_k} > T_j]$ In this case we have $\textsc{Alg} = O(2^k \widetilde{H_k})$. Moreover, $\widetilde{H_k} = O(H_n + \Sigma_n/2^k)$, in fact if $2^{s-1} < H_n \leq 2^s$, then the total height of sparse shelves is $\sum_{i \leq s} 2^i = 2^{s+1} = O(H_n)$. Furthermore, dense shelves are filled with constant density, therefore their total height is at most $O(\Sigma_n/2^k)$. Finally, we need to show that $2^k = O(W\sqrt{n})$. Thanks to the constant density packing of B_ℓ, we have $2^k H_j \sqrt{j} = O(2^\ell T_j) = O(\Sigma_j)$. We can upper bound the size of every piece p_i for $i \leq j$ with $W \times H_j$ and obtain $\Sigma_j \leq n \cdot WH_j$. Plugging it in the previous estimate and dividing both sides by $H_j\sqrt{n}$ we get $2^k = O(W\sqrt{n})$. Now we have $\textsc{Alg} = O(2^k \widetilde{H_k}) = O(2^k H_n + \Sigma_n) = O(WH_n\sqrt{n} + \Sigma_n) = \textsc{Opt} \cdot O(\sqrt{n})$. □

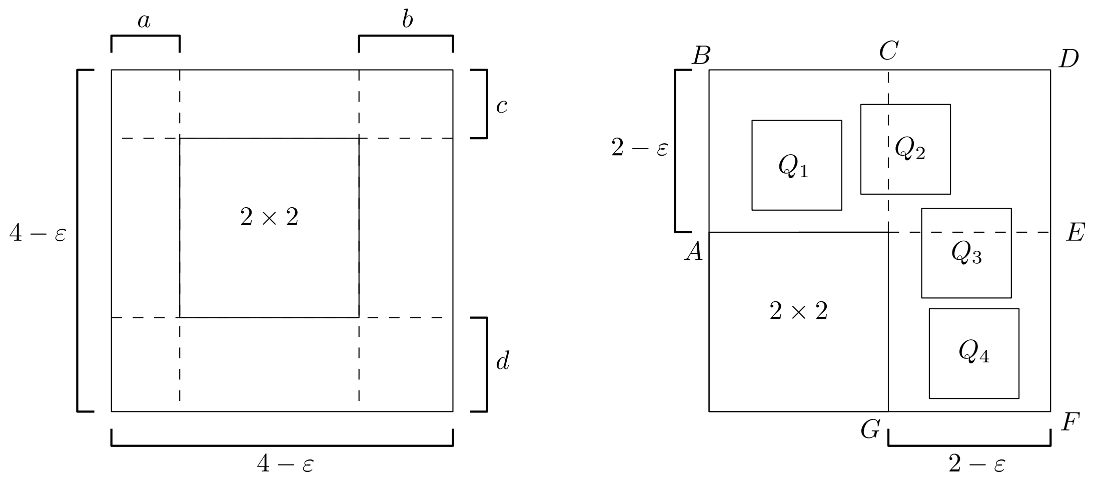


Figure 8: Left: A 2×2 square inside a bounding square having edges shorter than 4. Right: The 2×2 square has been dragged in the bottom left corner of the bounding square. Four 1×1 squares $Q_1, \ldots, Q_4$ are placed within the bounding square.

The algorithm DynBoxRot is obtained from DynBoxTrans with a slight modification: before processing any piece p_i we rotate it so that $w_i \leq h_i$. In this way, it still holds that $\text{Opt} = \Omega(\Sigma_n + WH_n)$ and the proof of Theorem 16 works also for the following.

Theorem 17. *The algorithm* DynBoxRot *has an absolute competitive ratio of $O(\sqrt{n})$ for the problem* AreaRotation *on a stream of n pieces.*

3.3 Bounded aspect ratio

In this section, we will consider the special case where the aspect ratio of all pieces is $\alpha = 1$, i.e., all the pieces are squares. Furthermore, we will measure the size of the packing as the area of the minimum axis-parallel bounding *square*, and we call the resulting problem SquareInSquareArea. Since we get a constant competitive ratio in this case, it follows that for other values of α and when allowing the bounding box to be a general rectangle, one can likewise achieve a constant competitive ratio. We first give a lower bound.

Lemma 18. *Consider any algorithm A for the problem* SquareInSquareArea. *Then the competitive ratio of A is at least* 16/9.

Proof. We first give A four 1×1 squares. Let the bounding square have size $\ell \times \ell$. If $\ell \geq 3$, the bounding square of the four 1×1 squares has size at least 3×3, while the optimal packing has size 2×2, which gives ratio at least 9/4. Otherwise, if $\ell < 3$, we give a 2×2 square and we will prove that the bounding square has size at least 4×4 while the optimal packing fits in a 3×3 square, so the ratio is at least 16/9.

Let us assume by contradiction that there exists a $(4 - \varepsilon) \times (4 - \varepsilon)$ bounding square containing both a 2×2 square and four 1×1 squares, with the additional hypothesis that the 1×1 squares fit in a $(3 - \delta) \times (3 - \delta)$ bounding box. We refer to notation in Figure 8 (left) and notice that we have $a < 1$ or $b < 1$, and analogously $c < 1$ or $d < 1$. Without loss of generality, we can assume $a, d < 1$. Hence, starting from the configuration in Figure 8 (left) we can drag the 2×2 square to the bottom left corner and obtain the configuration in Figure 8 (right), that still fulfill the hypotheses we assumed by contradiction.

From now on we employ the notation of Figure 8 (right). Let (x_i, y_i) be the coordinates of the bottom left corner of square Q_i. Stating that Q_i and Q_j are disjoint is equivalent to $\max\{|x_i - x_j|, |y_i - y_j|\} \geq 1$. Consider now the two rectangular regions $ABDE$ and $GCDF$: note that each of them can contain at most two squares. Indeed, given Q_i and Q_j completely contained in $ABDE$, it holds $|y_i - y_j| \leq 1 - \varepsilon$ thus $|x_i - x_j| \geq 1$. If three squares Q_1, Q_2, Q_3 are completely contain in $ABDE$ then we have, without loss of generality, $x_1 \leq x_2 - 1 \leq x_3 - 2$ and the minimal bounding square of Q_1, Q_2, Q_3 has size at least 3×3, that gives a contradiction. The same holds for $GCDF$.

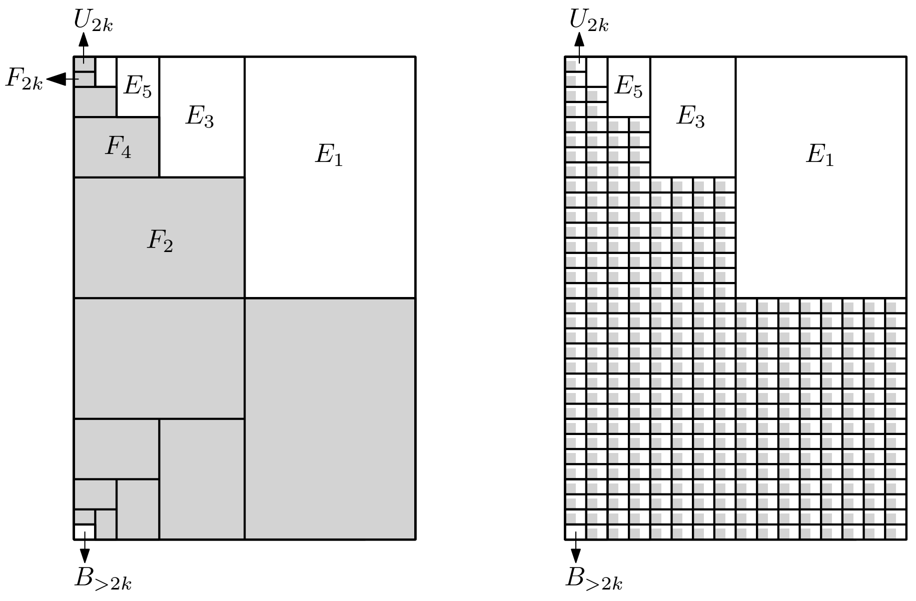


Figure 9: Left: A $2k$-packing. The grey bricks are non-empty and may have been split into smaller bricks. Right: The $2k$-packing produced by BrickTranslation when providing the algorithm with enough copies of the square S_k (the small grey squares), showing that the competitive ratio can be arbitrarily close to 6.

Finally, every Q_i is either fully contained in $ABDE$ or $GCDF$ hence, without loss of generality, we can assume that Q_1, Q_2 are contained in $ABDE$ and Q_3, Q_4 are contained in $GCDF$. This implies that $x_1 \leq x_2 - 1$ and $y_4 \leq y_3 - 1$, again without loss of generality. Observe that $x_2 \leq x_3 + 1 - \varepsilon$ and $y_3 \leq y_2 + 1 - \varepsilon$. Q_2 and Q_3 are disjoint, using the previous characterization we have two cases. First, $|x_2 - x_3| \geq 1$ and thanks to the observation above it cannot be $x_2 > x_3$, therefore we have $x_1 \leq x_2 - 1 \leq x_3 - 2$. Else, $|y_2 - y_3| \geq 1$ and thanks to the observation above we have $y_4 \leq y_3 - 1 \leq y_2 - 2$. In both cases that gives a contradiction since we cannot pack all Q_is in a $(3 - \delta) \times (3 - \delta)$ bounding square. □

We are now going to analyze the competitive ratio of the algorithm BrickTranslation (in fact, the algorithm BrickRotation has the exact same behavior when the pieces are squares). Note that a brick can never contain more than one piece. The algorithm is almost the same as the one described by Fekete and Hoffmann [13]. The slight difference is addressed in Section 2.2 and it is shown there that the behavior as described by Fekete and Hoffmann makes a worse algorithm for the problem PerimeterTranslation. However, even though the two algorithms will not always produce identical packings for the problem SquareInSquareArea, the analysis of the following theorem seems to hold for both versions, so for the problem SquareInSquareArea, the algorithms are equally good.

Theorem 19. *The algorithm* BrickTranslation *has a competitive ratio of* 6 *for* SquareInSquareArea. *The analysis is tight.*

Proof. Suppose a stream of squares have been packed by BrickTranslation, and let Alg be the area of the bounding square of the resulting packing. Let B_k be the largest elementary brick in which a square has been placed. Suppose without loss of generality that $k = 0$, so that B_k has size $1 \times 1/\sqrt{2}$ and $B_{\geq k}$, which contains all the packed squares, has size $1 \times \sqrt{2}$.

We now recursively define a type of packing that we call a $2k$-packing, for a non-negative integer k; see Figure 9 (left). As k increases, so do the requirements to a $2k$-packing, in the sense that a $(2k + 2)$-packing is also a $2k$-packing, but the other way is in general not the case. Define $F_0 := B_{\geq 1}$ and $U_0 := B_0$. A packing is a 0-packing if pieces have been placed in U_0 (the brick U_0 may or may not have been split in smaller bricks). Hence, the considered packing is a 0-packing by the assumption that a piece has been placed in B_0. Suppose that we have defined a $2k$-packing for some integer k. A $(2k + 2)$-packing is a $2k$-packing with the additional requirements that

- the brick U_{2k} has been split into $L := U_{2k} \dagger 1$ and $E_{2k+1} := U_{2k} \dagger 2$,
- the right brick E_{2k+1} is empty,
- the left brick L has been split into $F_{2k+2} := L \dagger 1$ and $U_{2k+2} := L \dagger 2$, and
- U_{2k+2} is non-empty, and thus also F_{2k+2} is non-empty.

The symbols U_j, E_j, F_j have been chosen such that the brick is a j-brick, i.e., the index tells the size of the brick.

Consider a $2k$-packing. It follows from the definition that along the top edge of $B_{\geq 0}$ from the right corner $(1, \sqrt{2})$ to the left corner $(0, \sqrt{2})$, we meet a sequence $E_1, E_3, \ldots, E_{2k-1}$ of empty bricks of decreasing size, and finally meet a non-empty brick U_{2k} which may have been split into smaller bricks.

Claim 20. *If the packing is a $2k$-packing and not a $(2k+2)$-packing, then* $\text{ALG}/\text{OPT} < 6$.

Since we pack a finite number of squares, the produced packing is a $2k$-packing but not a $(2k+2)$-packing for some sufficiently large k, so Claim 20 implies Theorem 19.

Let us now prove Claim 20. We first compute the area of the brick U_{2k} and the total areas of the bricks $F_0, F_2, \ldots, F_{2k}$, as these areas will be used often:

$$u_k := |U_{2k}| = 2^{-2k}/\sqrt{2} \tag{1}$$

$$f_k := \sum_{i=0}^{k} |F_{2i}| = \frac{2|B_{\geq 0}| - u_k}{3} = \frac{4 - 4^{-k}}{3\sqrt{2}}. \tag{2}$$

1) Suppose first that U_{2k} has not been split into smaller bricks. Then, since U_{2k} is non-empty by assumption, we know that U_{2k} contains a square S of size $s \times s$ where $s \in (s_l, s_h] = \left(\sqrt{2}^{-2k-2}, \sqrt{2}^{-2k-1}\right]$. Since the bricks $E_1, E_3, \ldots, E_{2k-1}$ are all empty, we get that the upper edge of the bounding square coincides with the upper edge of S, and we thus have

$$\text{ALG} \leq \text{ALG}(s) := (\sqrt{2} - (\sqrt{2}^{-2k-1} - s))^2.$$

The largest empty brick in the bricks F_{2i} can have size $|U_{2k}|/2$, so the total size of empty bricks in $F_0, F_2, \ldots, F_{2k}$ is $|U_{2k}|$. Moreover, the density of squares into bricks is at least $1/2\sqrt{2}$ and by (2), we get that

$$\text{OPT} \geq \text{OPT}(s) := \frac{f_k - u_k}{2\sqrt{2}} + s^2 = \frac{1 - 4^{-k}}{3} + s^2.$$

In the case that $k = 0$, we get

$$\frac{\text{ALG}}{\text{OPT}} \leq \frac{\text{ALG}(s)}{\text{OPT}(s)} = \frac{2s\sqrt{2} + 2s^2 + 1}{2s^2}.$$

A simple analysis shows that the fraction is largest when $s = s_l$, so we get the bound

$$\frac{\text{ALG}}{\text{OPT}} \leq \frac{2s_l\sqrt{2} + 2s_l^2 + 1}{2s_l^2} = 3 + 2\sqrt{2} < 5.83$$

Suppose now that $k > 0$. We divide into two cases of whether s is in the lower or the upper half of the range $(s_l, s_h]$. For the lower half, that is, $s \in (s_l, \frac{s_l+s_h}{2}]$, we get

$$\frac{\text{ALG}}{\text{OPT}} \leq \frac{\text{ALG}(\frac{s_l+s_h}{2})}{\text{OPT}(s_l)} = \frac{96 \cdot 4^k + (24\sqrt{2} - 48) \cdot 2^k - 6\sqrt{2} + 9}{16 \cdot 4^k - 4}.$$

It is straightforward to check that $(24\sqrt{2} - 48) \cdot 2^k - 6\sqrt{2} + 9 < 6 \cdot (-4)$ for all $k \geq 1$, so it follows that the ratio is less than 6.

For the upper half, that is, $s \in [\frac{s_l+s_h}{2}, s_h]$, we get

$$\frac{\text{ALG}}{\text{OPT}} \leq \frac{\text{ALG}(s_h)}{\text{OPT}(\frac{s_l+s_h}{2})} = \frac{96 \cdot 4^k}{16 \cdot 4^k + 6\sqrt{2} - 7}.$$

As $6\sqrt{2} - 7 > 0$, the ratio is less than 6.

2) We now assume that U_{2k} has been split into a L and E_{2k+1}, which are the left and right halfs of U_{2k}, respectively.

   2.1) We first suppose that E_{2k+1} is not empty. This implies that there is no empty $(2k+1)$-brick in $F_0, F_2, \ldots, F_{2k}, U_{2k}$. Hence, each empty brick in the bricks $F_0, F_2, \ldots, F_{2k}, U_{2k}$ is a $(2k+2)$-brick or smaller, so these empty bricks have total size at most $u_k/2$. We then get

$$\textsc{Opt} \geq \frac{f_k + u_k - u_k/2}{2\sqrt{2}} = \frac{1}{3} + \frac{4^{-k}}{24} > \frac{1}{3}.$$

   Since $\textsc{Alg} \leq 2$, it follows that $\frac{\textsc{Alg}}{\textsc{Opt}} < 6$.

   2.2) We now suppose that E_{2k+1} is empty.

   2.2.1) Suppose now that L has not been split into smaller bricks. Then L contains a square S of size $s \times s$ for $s \in (s_l, s_h] = \left(\sqrt{2}^{-2k-3}, \sqrt{2}^{-2k-2}\right]$. As in case 1, we get

$$\textsc{Alg} \leq \textsc{Alg}(s) := (\sqrt{2} - (\sqrt{2}^{-2k-1} - s))^2.$$

   Note that there is no empty $(2k+1)$-brick in the bricks $F_0, F_2, \ldots, F_{2k}$, so these bricks contain a total area of at most $u_k/2$ empty bricks. We then get

$$\textsc{Opt} \geq \textsc{Opt}(s) := \frac{f_k - u_k/2}{2\sqrt{2}} + s^2.$$

   We then get the bound

$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{\textsc{Alg}(s_h)}{\textsc{Opt}(s_l)} = \frac{24 \cdot 4^k + (12\sqrt{2} - 24) \cdot 2^k - 6\sqrt{2} + 9}{4 \cdot 4^k - 1}.$$

   Here, it is straightforward to verify that $(12\sqrt{2} - 24) \cdot 2^k - 6\sqrt{2} + 9 < 6 \cdot (-1)$ for all $k \geq 0$, and hence the ratio is less than 6.

   2.2.2) We now assume that L has been split into F_{2k+2} and U_{2k+2}, which are the bottom and top parts, respectively.

   2.2.2.1) Suppose that U_{2k+2} is empty. Since also $E_1, E_3, \ldots, E_{2k+1}$ are empty, we get that $\textsc{Alg} \leq (\sqrt{2} - \sqrt{2}^{-2k-1}/2)^2$.
   Note that each empty bricks in the bricks $F_0, F_2, \ldots, F_{2k+2}$ can have size at most $u_k/8$, so the total size of the empty bricks is at most $u_k/4 = u_{k+1}$, and we get

$$\textsc{Opt} \geq \frac{f_{k+1} - u_{k+1}}{2\sqrt{2}}.$$

   We therefore get

$$\frac{\textsc{Alg}}{\textsc{Opt}} \leq \frac{48 \cdot 4^k - 24 \cdot 2^k + 3}{8 \cdot 2^{2\,k} - 2}.$$

   Here, it is straightforward to check that $-24 \cdot 2^k + 3 < 6 \cdot (-2)$ for all $k \geq 0$, so the ratio is less than 6.

   2.2.2.2) We are finally left with the case that U_{2k+2} is not empty. But then all the requirements are satisfied for the packing to be a $(2k+2)$-packing.

We now observe that the analysis is tight. To this end, we show that for any given k and a small $\varepsilon > 0$, we can force the algorithm to produce a $2k$-packing, such that as $k \longrightarrow \infty$ and $\varepsilon \longrightarrow 0$, the ratio $\frac{\textsc{Alg}}{\Sigma}$ tends to 6, where Σ is the total area of the packed squares. Let $\varepsilon_k := \varepsilon\sqrt{2}^{-k}$, $\ell_k := \sqrt{2}^{-k}/2 + \varepsilon_k$, and let S_k be a square of size $\ell_k \times \ell_k$. We now feed the algorithm with copies of S_k. This will eventually result in a $2k$-packing, where each non-empty brick is a $2k$-brick; see Figure 9 (right). Let n_k be the number needed to produce the $2k$-packing. The density in each non-empty brick is $\rho_\varepsilon := \frac{|S_k|}{|B_{2k}|}$. As $\varepsilon \longrightarrow 0$, we get that $\rho_\varepsilon \longrightarrow \frac{1}{2\sqrt{2}}$. As $k \longrightarrow \infty$, the area of non-empty bricks converges to $\frac{2|B_{\leq 0}|}{3} = \frac{2\sqrt{2}}{3}$. Hence, we have $\Sigma \longrightarrow \frac{1}{2\sqrt{2}} \cdot \frac{2\sqrt{2}}{3} = \frac{1}{3}$. We then get $\frac{\textsc{Alg}}{\Sigma} \longrightarrow \frac{2}{1/3} = 6$. Furthermore, the optimal packing of the squares is to place them so that their bounding box is a square of size $\lceil\sqrt{n_k}\rceil\ell_k \times \lceil\sqrt{n_k}\rceil\ell_k$. As $k \longrightarrow \infty$, we then have $\frac{\Sigma}{\textsc{Opt}} \longrightarrow 1$. Hence, we have $\frac{\textsc{Alg}}{\textsc{Opt}} \longrightarrow 6$. $\square$

3.4 More lower bounds when edges are long

We already saw in Corollary 14 that as a function of n, the competitive ratio of an algorithm for AreaTranslation or AreaRotation must be at least $\Omega(\sqrt{n})$, even when all edges have length 1. In this section, we give lower bounds in terms of Opt for the same case. Note that the assumption that the edges are long is needed for these bounds to be matched by actual algorithms, since Corollary 13 states that without the assumption, the competitive ratio cannot be bounded as a function of Opt.

Theorem 21. *Consider any algorithm A for the problem* AreaTranslation *with the restriction that all edges of the given rectangles have length at least* 1. *If A has an asymptotic competitive ratio $f(\textsc{Opt})$ as a function of* Opt, *then $f(\textsc{Opt}) = \Omega(\sqrt{\textsc{Opt}})$.*

Remark 22. Note that when the edges are long, $\Omega(\sqrt{\textsc{Opt}}) = \Omega(\sqrt{n})$, so this bound is stronger than the $\Omega(\sqrt{n})$ bound of Corollary 14.

Proof of Theorem 21. For any $n \in \mathbb{N}$, we do as follows. We first provide A with n^2 unit squares. Let the bounding box of the produced packing of these squares have size $a \times b$. Assume without loss of generality that $a \leq b$, so that $b \geq n$. We now give A the rectangle $n^2 \times 1$. The optimal offline solution to this set of rectangles has a bounding box of size $n^2 \times 2$. The packing produced by A has a bounding box of size at least $n^2 \times n = \Omega(\sqrt{\textsc{Opt}}) \cdot \textsc{Opt}$. □

Theorem 23. *Consider any algorithm A for the problem* AreaRotation *with the restriction that all edges of the given rectangles have length at least* 1. *If A has a competitive ratio $f(\textsc{Opt})$ as a function of* Opt, *then $f(\textsc{Opt}) = \Omega(\sqrt[4]{\textsc{Opt}})$.*

Proof. For any $n \in \mathbb{N}$, we do as follows. We first provide A with n^2 unit squares. Let the bounding box of the produced packing of these squares have size $a \times b$. Assume without loss of generality that $a \leq b$. If $a \geq n^{1/2}$, we give A the rectangle $1 \times n^2$. Otherwise, we have $b > n^{3/2}$, and then we give A the square $n \times n$. In either case, there is an optimal offline solution of area $2n^2$, but the bounding box of the packing produced by A has area at least $n^{5/2} = \Omega(\sqrt[4]{\textsc{Opt}}) \cdot \textsc{Opt}$. □

3.5 Algorithms when edges are long

In this section, we describe algorithms that match lower bounds of Section 3.4. We analyze these algorithms under the assumption that we feed them with rectangles with edges of length at least 1 (of course, any other positive constant will also work), but we require no bound on the aspect ratio. Under this assumption, we observe that DynBoxTrans has absolute competitive ratio $O(\sqrt{\textsc{Opt}})$ for AreaTranslation. We then describe the algorithm DynBoxRot$_{\sqrt[4]{\textsc{Opt}}}$, which we prove to have absolute competitive ratio $O(\sqrt[4]{\textsc{Opt}})$ for AreaRotation. By Theorems 21 and 23, both algorithms are optimal to within a constant factor.

In previous sections we proved lower bounds of $\Omega(\sqrt{n})$ and $\Omega(\sqrt[4]{\textsc{Opt}})$ for AreaRotation. They can be summarized stating that AreaRotation has a competitive ratio of $\Omega(\max\{\sqrt{n}, \sqrt[4]{\textsc{Opt}}\})$. The last theorem of this section, describes the algorithm DynBoxRot$_{\sqrt{n}\wedge\sqrt[4]{\textsc{Opt}}}$ that simultaneously matches both lower bounds achieving a competitive ratio of $O(\min\{\sqrt{n}, \sqrt[4]{\textsc{Opt}}\}$. At a first sight it may seem that this algorithm contradicts the lower bound of $\Omega(\max\{\sqrt{n}, \sqrt[4]{\textsc{Opt}}\})$; however this simply proves that the *edge cases* that have a competitive ratio of at least $\Omega(\sqrt[4]{\textsc{Opt}})$ must satisfy $\textsc{Opt} = O(n^2)$. Likewise, those for which the competitive ratio is at least $\Omega(\sqrt{n})$ satisfy $n = O(\sqrt{\textsc{Opt}})$.

Translations only Under the long edge assumption, we have $n \leq \textsc{Opt}$. Therefore, DynBoxTrans achieves a competitive ratio of $O(\sqrt{n}) = O(\sqrt{\textsc{Opt}})$ for AreaTranslation and matches the bound stated in Theorem 21.

Rotations allowed Now we tackle the AreaRotation problem and describe the algorithm DynBoxRot$_{\sqrt[4]{\textsc{Opt}}}$. We define the threshold function $T_j = \Sigma_j^{3/4} + 7H_j$, where $H_j = \max_{i=1,\ldots,j} h_i$ and Σ_j is the total area of pieces $p_1, \ldots, p_j$. DynBoxRot$_{\sqrt[4]{\textsc{Opt}}}$ is obtained by running DynBoxRot, as described in Section 3.2, employing this new threshold T_j.

Theorem 24. *The algorithm* DynBoxRot$_{\sqrt[4]{\text{Opt}}}$ *has an absolute competitive ratio of* $O(\sqrt[4]{\text{Opt}})$ *for the problem* AreaRotation, *where* Opt *is the area of the optimal offline packing.*

Proof. This proof is similar to the one of Theorem 16. Define $W := \max_{i=1,\dots,n} w_i$. Recall that in DynBoxRot we preprocess every piece p rotating it so the $w_p \le h_p$, hence $W \le \sqrt{\Sigma_n}$. Let B_k be the last active box, so that we can enclose all the pieces in a bounding box of size $2^{k+1} \times T_n$, and bound the area returned by the algorithm as $\text{Alg} = O(2^k H_n + 2^k \Sigma_n^{3/4})$. On the other hand we are able to bound the optimal offline packing as $\text{Opt} = \Omega(\Sigma_n + W H_n)$.

If the active box never changed, then we have $2^k < 2W$ that implies $\text{Alg} = O(WH_n + \Sigma_n^{5/4}) = \text{Opt} \cdot O(\sqrt[4]{\text{Opt}})$. Otherwise, let B_ℓ be the last active box before B_k, and p_j be the first piece put in B_k. Here we have two cases.
Case (1) $[w_j > 2^\ell]$ In this case we have $2^k < 2W$ that implies $\text{Alg} = O(WH_n + \Sigma_n^{5/4}) = \text{Opt} \cdot O(\sqrt[4]{\text{Opt}})$.
Case (2) $[w_j \le 2^\ell]$ In this case we have $k = \ell + 1$. Denote with $\widetilde{H_i}$ the total height of shelves in B_i. Then we have $\widetilde{H_\ell} \ge T_j - H_j = \Sigma_j^{3/4} + 6H_j$, otherwise we could pack p_j in B_ℓ. Thus, we can apply Lemma 15 and conclude that the box B_ℓ of size $2^\ell \times T_j$ is filled with constant density. Here we have two cases.
Case (2.1) $[\widetilde{H_k} \le T_j]$ In this case we have $\text{Alg} = O(2^k T_j)$ and, thanks to the constant density packing of B_ℓ we have $\Sigma_j = \Theta(2^\ell \widetilde{H_\ell}) = \Theta(2^k T_j)$. Since $\text{Opt} \ge \Sigma_j$, we get $\text{Alg} = O(\text{Opt})$.
Case (2.2) $[\widetilde{H_k} > T_j]$ In this case we have $\text{Alg} = O(2^k \widetilde{H_k})$. Moreover, $\widetilde{H_k} = O(H_n + \Sigma_n / 2^k)$, in fact if $2^{s-1} < H_n \le 2^s$, then the total height of sparse shelves is $\sum_{i \le s} 2^i = 2^{s+1} = O(H_n)$. Furthermore, dense shelves are filled with constant density, therefore their total height is at most $O(\Sigma_n/2^k)$. Finally, we need to show that $2^k = O(\sqrt[4]{\Sigma_n})$. Thanks to the constant density packing of B_ℓ, we have $2^k \Sigma_j^{3/4} = O(2^\ell T_j) = O(\Sigma_j)$. Dividing both sides by $\Sigma_j^{3/4}$ we get $2^k = O(\Sigma_j^{1/4})$. In the end notice that, thanks to the long edge hypotheses $H_n \le \Sigma_n$ and we have $\text{Alg} = O(2^k \widetilde{H_k}) = O(2^k H_n + \Sigma_n) = O(\Sigma_n^{5/4}) = \text{Opt} \cdot O(\sqrt[4]{\text{Opt}})$. □

So far we managed to match the competitive ratio lower bounds of $\Omega(\sqrt{n})$ and $\Omega(\sqrt[4]{\text{Opt}})$ employing two different algorithms: DynBoxRot and DynBoxRot$_{\sqrt[4]{\text{Opt}}}$. A natural question is whether is it possible to match the performance of these algorithms simultaneously, having an algorithm that achieves a competitive ratio of $O(\min\{\sqrt{n}, \sqrt[4]{\text{Opt}}\})$. We give an affirmative answer by describing the algorithm DynBoxRot$_{\sqrt{n} \wedge \sqrt[4]{\text{Opt}}}$.

Again, we employ the same scheme of DynBoxRot with a different threshold function. This time the definition of T_j is slightly more involved. First define

$$\widetilde{T_j} = \begin{cases} \Sigma_j^{3/4} + 7H_j, & \text{if } \Sigma_j < j^2 \\ H_j\sqrt{n} + 7H_j, & \text{otherwise.} \end{cases}$$

Later we will write $\widetilde{T_j}$ as $\widetilde{T_j} = \mathbb{1}_{\{\Sigma_j < j^2\}} \cdot \Sigma_j^{3/4} + \mathbb{1}_{\{\Sigma_j \ge j^2\}} \cdot H_j\sqrt{n} + 7H_j$. We now define

$$T_j = \begin{cases} 0, & \text{if } j = 0 \\ \max\left\{T_{j-1}, \widetilde{T_j}\right\}, & \text{if } j \ge 1. \end{cases}$$

This two-step definition is necessary for a correct implementation of the algorithm because we must guarantee that T_j does not decrease.

Theorem 25. *When used on the problem* AreaRotation, *the algorithm* DynBoxRot$_{\sqrt{n} \wedge \sqrt[4]{\text{Opt}}}$ *has an absolute competitive ratio of* $O(\min\{\sqrt{n}, \sqrt[4]{\text{Opt}}\})$, *where* Opt *is the area of the optimal offline packing and n is the total number of pieces in the stream.*

Proof. Again, we define $W := \max_{i=1,\dots,n} w_i$. Recall that in DynBoxRot we preprocess every piece p rotating it so the $w_p \le h_p$, hence $W \le \sqrt{\Sigma_n}$. Let B_k be the last active box, so that we can enclose all the pieces in a bounding box of size $2^{k+1} \times T_n$. There exists a $n' \le n$ such that $T_n = \widetilde{T}_{n'}$. We can bound the area returned by the algorithm as

$$\text{Alg} = O\left(2^k \widetilde{T}_{n'}\right) = O\left(2^k H_{n'} + \mathbb{1}_{\{\Sigma_{n'} < n'^2\}} \cdot 2^k \Sigma_{n'}^{3/4} + \mathbb{1}_{\{\Sigma_{n'} \ge n'^2\}} \cdot 2^k H_{n'} \sqrt{n'}\right).$$

We bound the optimal offline packing as $\text{Opt} = \Omega(\Sigma_n + WH_n)$. If the active box never changed, then we have $2^k < 2W$ that implies

$$\begin{aligned}
ALG &= O\left(WH_n + \mathbb{1}_{\{\Sigma_{n'}<n'^2\}} \cdot W\Sigma_{n'}^{3/4} + \mathbb{1}_{\{\Sigma_{n'}\geq n'^2\}} \cdot WH_{n'}\sqrt{n'}\right) \\
&= O\left(WH_n + \mathbb{1}_{\{\Sigma_{n'}<n'^2\}} \cdot \Sigma_n\sqrt[4]{\Sigma_{n'}} + \mathbb{1}_{\{\Sigma_{n'}\geq n'^2\}} \cdot WH_n\sqrt{n'}\right) \\
&\leq \text{Opt} \cdot O\left(\min\left\{\sqrt[4]{\Sigma_{n'}}, \sqrt{n'}\right\}\right) = \text{Opt} \cdot O\left(\min\left\{\sqrt[4]{\text{Opt}}, \sqrt{n}\right\}\right).
\end{aligned}$$

Otherwise, let B_ℓ be the last active box before B_k, and p_j be the first piece put in B_k. Here we have two cases.
Case (1) $[w_j > 2^\ell]$ In this case we have, again, $2^k < 2W$ and we use the same argument employed above.
Case (2) $[w_j \leq 2^\ell]$ In this case we have $k = \ell + 1$. Denote with $\widetilde{H_i}$ the total height of shelves in B_i. Then we have $\widetilde{H_\ell} \geq T_j - H_j \geq \widetilde{T}_j - H_j \geq 6H_j$, otherwise we could pack p_j in B_ℓ. Thus, we can apply Lemma 15 and conclude that the box B_ℓ of size $2^\ell \times T_j$ is filled with constant density. Here we have two cases.
Case (2.1) $[\widetilde{H_k} \leq T_j]$ In this case we have $\text{Alg} = O(2^k T_j)$ and, thanks to the constant density packing of B_ℓ we have $\Sigma_j = \Theta(2^\ell \widetilde{H_\ell}) = \Theta(2^k T_j)$. Since $\text{Opt} \geq \Sigma_j$, we get $\text{Alg} = O(\text{Opt})$.
Case (2.2) $[\widetilde{H_k} > T_j]$ In this case we still have $\text{Alg} = O(2^k \widetilde{H_k})$. Moreover, $\widetilde{H_k} = O(H_n + \Sigma_n/2^k)$, in fact if $2^{s-1} < H_n \leq 2^s$, then the total height of sparse shelves is $\sum_{i\leq s} 2^i = 2^{s+1} = O(H_n)$. Furthermore, dense shelves are filled with constant density, therefore their total height is at most $O(\Sigma_n/2^k)$.

Finally, we need to show that $2^k = O(\min\{\sqrt[4]{\Sigma_n}, \sqrt{n}\})$. Let $T_j = \widetilde{T}_{j'}$, we have two cases.
Case (2.2.1) $[\Sigma_{j'} < j'^2]$ We have $\widetilde{T}_{j'} \geq \Sigma_{j'}^{3/4}$. And thanks to the constant density packing of B_ℓ, we have also $2^k \Sigma_{j'}^{3/4} = O(2^\ell T_j) = O(\Sigma_{j'})$. Dividing both sides by $\Sigma_{j'}^{3/4}$ we get $2^k = O(\sqrt[4]{\Sigma_{j'}})$.
Case (2.2.2) $[\Sigma_{j'} \geq j'^2]$ In this case we have $\widetilde{T}_{j'} \geq H_{j'}\sqrt{j'}$. Using the constant density argument we get $2^k H_{j'}\sqrt{j'} = O(2^k \widetilde{T}_{j'}) = O(\Sigma_{j'}) \leq O(j' \cdot WH_{j'})$. Dividing both sides by $H_{j'}\sqrt{j'}$ we obtain $2^k = O(W\sqrt{j'})$. Therefore, we have

$$2^k = \begin{cases} O(\sqrt[4]{\Sigma_{j'}}) & \text{if } \Sigma_{j'} < j'^2 \\ W\sqrt{j'} & \text{otherwise.} \end{cases}$$

In the end notice that, thanks to the long edge hypotheses $H_n \leq \Sigma_n$, thus

$$\begin{aligned}
\text{Alg} &= O\left(2^k \widetilde{H_k}\right) = O\left(2^k H_n + \Sigma_n\right) \\
&= O\left(\mathbb{1}_{\{\Sigma_{j'}<j'^2\}} \cdot H_n\sqrt[4]{\Sigma_{j'}} + \mathbb{1}_{\{\Sigma_{j'}\geq j'^2\}} \cdot WH_n\sqrt{j'} + \Sigma_n\right) \\
&\leq \text{Opt} \cdot O\left(\min\left\{\sqrt[4]{\Sigma_{j'}}, \sqrt{j'}\right\}\right) = \text{Opt} \cdot O\left(\min\left\{\sqrt[4]{\text{Opt}}, \sqrt{n}\right\}\right).
\end{aligned}$$

□

4 Further questions

It is natural to consider problems where the given pieces are more general, such as convex polygons. Here, we may allow the pieces to be rotated by arbitrary angles. In that case, it follows from the technique described by Alt [2] that one can obtain a constant competitive ratio for computing a packing with a minimum perimeter bounding box: For each new piece, we rotate the piece so that a diameter of the piece is horizontal. We then use the algorithm BrickRotation to pack the bounding boxes of the pieces. Since the area of each piece is at least half of the area of its bounding box, the density of the produced packing is at least half of the density of the packing of the bounding boxes. This results in an increase of the competitive ratio by a factor of at most $\sqrt{2}$.

For the problem of minimizing the perimeter of the bounding box (or convex hull) with convex polygons as pieces and only translations allowed, we do not know if it is possible to get a competitive ratio of $O(1)$, and this seems to be a very interesting question for future research. In order to design such an algorithm, it would be sufficient to show that for some constants $\delta > 0$ and $\Sigma > 0$, there is an online algorithm that packs any stream of convex polygons of diameter at most δ and total area at most Σ into the unit square, which is in itself an interesting problem. The three-dimensional version of this question has a negative answer, even for offline algorithms: Alt, Cheong, Park, and Scharf [3] showed that for any $n \in \mathbb{N}$, there exists a finite number of 2D unit disks embedded in 3D that cannot all be packed by translation in a cube with edges of length n.

References

[1] Hee-Kap Ahn and Otfried Cheong. Aligning two convex figures to minimize area or perimeter. *Algorithmica*, 62(1-2):464–479, 2012.

[2] Helmut Alt. Computational aspects of packing problems. *Bulletin of the EATCS*, 118, 2016.

[3] Helmut Alt, Otfried Cheong, Ji-won Park, and Nadja Scharf. Packing 2D disks into a 3D container. In *International Workshop on Algorithms and Computation (WALCOM 2019)*, pages 369–380, 2019.

[4] Helmut Alt, Mark de Berg, and Christian Knauer. Approximating minimum-area rectangular and convex containers for packing convex polygons. In *23rd Annual European Symposium on Algorithms (ESA 2015)*, pages 25–34, 2015.

[5] Helmut Alt and Ferran Hurtado. Packing convex polygons into rectangular boxes. In *18th Japanese Conference on Discrete and Computational Geometry (JCDCGG 2000)*, pages 67–80, 2000.

[6] Brenda S. Baker and Jerald S. Schwarz. Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing*, 12(3):508–525, 1983.

[7] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 2005.

[8] Brian Brubach. Improved bound for online square-into-square packing. In *12th International Workshop on Approximation and Online Algorithms (WAOA 2014)*, pages 47–58, 2014.

[9] Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

[10] Fan Chung and Ron Graham. Efficient packings of unit squares in a large square. *Discrete & Computational Geometry*, 2019.

[11] János Csirik and Gerhard J. Woeginger. On-line packing and covering problems. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 147–177. Springer, 1998.

[12] Paul Erdős and Ron Graham. On packing squares with equal squares. *Journal of Combinatorial Theory, Series A*, 19(1):119–123, 1975.

[13] Sándor P. Fekete and Hella-Franziska Hoffmann. Online square-into-square packing. *Algorithmica*, 77(3):867–901, 2017.

[14] Amos Fiat and Gerhard J. Woeginger. Competitive analysis of algorithms. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 1–12. Springer, 1998.

[15] Janusz Januszewski and Marek Lassak. On-line packing sequences of cubes in the unit cube. *Geometriae Dedicata*, 67(3):285–293, 1997.

[16] Marek Lassak. On-line potato-sack algorithm efficient for packing into small boxes. *Periodica Mathematica Hungarica*, 34(1-2):105–110, 1997.

[17] Hyun-Chan Lee and Tony C. Woo. Determining in linear time the minimum area convex hull of two polygons. *IIE Transactions*, 20(4):338–345, 1988.

[18] Boris D. Lubachevsky and Ronald L. Graham. Dense packings of congruent circles in rectangles with a variable aspect ratio. In Boris Aronov, Saugata Basu, János Pach, and Micha Sharir, editors, *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, pages 633–650. 2003.

[19] Boris D. Lubachevsky and Ronald L. Graham. Minimum perimeter rectangles that enclose congruent non-overlapping circles. *Discrete Mathematics*, 309(8):1947–1962, 2009.

[20] Victor J. Milenkovic. Translational polygon containment and minimal enclosure using linear programming based restriction. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing (STOC 1996)*, pages 109–118, 1996.

[21] Victor J. Milenkovic. Rotational polygon containment and minimum enclosure using only robust 2D constructions. *Computational Geometry*, 13(1):3–19, 1999.

[22] Victor J. Milenkovic and Karen Daniels. Translational polygon containment and minimal enclosure using mathematical programming. *International Transactions in Operational Research*, 6(5):525–554, 1999.

[23] Dongwoo Park, Sang Won Bae, Helmut Alt, and Hee-Kap Ahn. Bundling three convex polygons to minimize area or perimeter. *Computational Geometry*, 51:1–14, 2016.

[24] E. Specht. High density packings of equal circles in rectangles with variable aspect ratio. *Computers & Operations Research*, 40(1):58 –69, 2013.

[25] Rob van Stee. SIGACT news online algorithms column 20: the power of harmony. *SIGACT News*, 43(2):127–136, 2012.

[26] Rob van Stee. SIGACT news online algorithms column 26: Bin packing in multiple dimensions. *SIGACT News*, 46(2):105–112, 2015.

Appendix D

SODA: Online Sorting and Translational Packing of Convex Polygons

Online Sorting and Translational Packing of Convex Polygons

Anders Aamand*[1], Mikkel Abrahamsen*[2], Lorenzo Beretta*[2], and Linda Kleist*[3]

[1]MIT, aamand@mit.edu
[2]BARC, University of Copenhagen, {miab,beretta}@di.ku.dk
[3]Technische Universität Braunschweig, kleist@ibr.cs.tu-bs.de

December 2021

Abstract

We investigate various online packing problems in which convex polygons arrive one by one and have to be placed irrevocably into a container before the next piece is revealed; the pieces must not be rotated, but only translated. The aim is to minimize the used space depending on the specific problem at hand, e.g., the strip length in strip packing, the number of bins in bin packing, etc.

We draw interesting connections to the following online sorting problem ONLINE-SORTING$[\gamma, n]$: We receive a stream of real numbers $s_1, \ldots, s_n$, $s_i \in [0,1]$, one by one. Each real must be placed in an array A with γn initially empty cells without knowing the subsequent reals. The goal is to minimize the sum of differences of consecutive reals in A. The offline optimum is to place the reals in sorted order so the cost is at most 1. We show that for any Δ-competitive online algorithm of ONLINE-SORTING$[\gamma, n]$, it holds that $\gamma\Delta \in \Omega(\log n / \log\log n)$.

We use this lower bound to answer several fundamental questions about packing. Specifically, we prove the non-existence of competitive algorithms for various online translational packing problems of convex polygons, among them strip packing, bin packing and perimeter packing. These results remain true even if the diameter of all pieces is bounded by any small constant $\delta > 0$, and even in the asymptotic case, i.e., for arbitrarily large values of OPT. This also implies that there exists no online algorithm that can pack all streams of pieces of diameter and total area at most δ into the unit square, i.e., the critical packing density is 0. These results are in contrast to the case when the pieces are restricted to rectangles, for which competitive algorithms are known. Likewise, the offline versions of packing convex polygons have constant factor approximation algorithms. These offline algorithms sort the convex polygons by the slope of their spine segment so that they form a fan-like pattern. In essence, our result shows that the impossibility of solving the online sorting problem implies the non-existence of a competitive algorithm, as it prevents arranging the pieces in a fan-like pattern.

On the positive side, we present an algorithm with competitive ratio $O(n^{0.59})$ for online translational strip packing of convex polygons. This beats the trivial n-competitive algorithm that places each new piece as far into the strip as possible.

For ONLINE-SORTING$[1, n]$, i.e., if A has n cells, we show that any online algorithm has a competitive ratio in $\Omega(\sqrt{n})$, and describe an algorithm with competitive ratio $O(\sqrt{n})$. This can be seen as an asymptotically tight analysis of an online variant of the traveling salesperson problem on the real line. In the case of ONLINE-SORTING$[C, n]$ for any constant $C > 1$, we present an $O(2^{O(\sqrt{\log n \log\log n})})$-competitive algorithm.

* Anders Aamand is funded by a DFF-International Postdoc Grant from the Independent Research Fund Denmark. Mikkel Abrahamsen and Lorenzo Beretta are part of Basic Algorithms Research Copenhagen (BARC), supported by the VILLUM Foundation grant 16582. Lorenzo Beretta receives funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 801199. Linda Kleist was able to visit BARC by the support of a postdoc fellowship of the German Academic Exchange Service (DAAD).

1 Introduction

Packing problems are omnipresent in our daily lives and likewise appear in many large-scale industries. For instance, two-dimensional versions of packing arise when a given set of pieces have to be cut out from a large piece of material such that waste is minimized. This is relevant in clothing production where pieces are cut out from a strip of fabric, and similarly in leather, glass, wood, and sheet metal cutting. In these settings, it is often important that the inherent structure of the host material (grain of fabric, patterns, etc.) is respected, i.e., the pieces should not be arbitrarily rotated, but merely translated. In some applications, the objects appear in an *online* fashion, i.e., the pieces appear one after the other, and each of them must be placed before the next one is known. This is in contrast to *offline* problems, where all the pieces are known in advance. Problems related to packing were some of the first for which online algorithms were described and analyzed. Indeed, the first use of the terms "online" and "offline" in the context of approximation algorithms was in the early 1970s and used for algorithms for bin-packing problems [20]. Most existing research on packing, and all research on online translational packing that we are aware of, is concerned with axis-parallel rectangular pieces.

In this paper, we study online translational packing of convex polygons. The pieces arrive one by one and have to be placed irrevocably into a horizontal strip (or into bins, a square, the plane) before the next piece is revealed, and only translations of the pieces are allowed. The aim is to minimize the used space depending on the specific problem at hand, e.g., the used length of the strip, the number of bins, etc.

To develop lower bounds for these packing problems, we introduce the problem ONLINE-SORTING$[\gamma, n]$ which we believe to be of independent interest. In this problem, we have an empty array A with γn cells, $\gamma \geq 1$, and receive a stream of real numbers $s_1, \ldots, s_n$, $s_i \in [0,1]$. Each real has to be placed into an empty cell of A before the next real is known. The goal is to minimize the sum of differences of consecutive reals in A. The offline optimum is obtained by placing the reals in sorted order in some n cells of A. We show that ONLINE-SORTING does not allow for constant factor competitive online algorithms.

Theorem 1. *Suppose that $\gamma, \Delta \geq 1$ are such that* ONLINE-SORTING$[\gamma, n]$ *admits a Δ-competitive algorithm, then $\gamma\Delta = \Omega(\log n / \log\log n)$.*

We then use this insight to show that various packing problems do not allow for constant factor asymptotically competitive online algorithms. In STRIP-PACKING, we have a horizontal strip of height 1 which is bounded to the left by a vertical segment and unbounded to the right. The goal is to place the pieces so that we use a part of the strip of minimum length. In BIN-PACKING, the pieces have to be placed in unit squares, and the goal is to use a minimum number of these squares. In PERIMETER-PACKING, we can place the pieces anywhere in the plane, and the goal is to minimize the perimeter of their convex hull. In SQUARE-PACKING$[\delta]$, we receive a stream of pieces with diameter at most δ and total area at most δ, and the goal is to place them in a unit square. For more background on each of these packing problems and their relation to previous work, we refer to Section 1.2.

Theorem 2. *The following holds, where n is the number of pieces:*

(a) STRIP-PACKING *does not allow for a competitive online algorithm, even if all pieces have diameter at most δ for any constant $\delta > 0$. In particular, the competitive ratio of any algorithm is $\Omega(\sqrt{\log n / \log\log n})$.*

(b) BIN-PACKING *does not allow for a competitive online algorithm, even if all pieces have diameter at most δ for any constant $\delta > 0$. In particular, the competitive ratio of any algorithm is $\Omega(\sqrt{\log n / \log\log n})$.*

(c) PERIMETER-PACKING *does not allow for a competitive online algorithm, even if all pieces have diameter at most δ for any constant $\delta > 0$. In particular, the competitive ratio of any algorithm is $\Omega(\sqrt[4]{\log n / \log\log n})$.*

(d) SQUARE-PACKING$[\delta]$ *does not allow for an online algorithm for any $\delta \in (0,1]$. In particular, for any algorithm and infinitely many n, there exists a stream of n pieces of total area $O(\sqrt{\log\log n / \log n})$ that the algorithm cannot pack in the unit square.*

Here, (a) and (b) even hold in the asymptotic sense, i.e., if we restrict ourselves to instances with offline optimal cost at least C, for any constant $C > 0$.

On the positive side, we present online algorithms for both online sorting and strip packing. For ONLINE-SORTING$[\gamma, n]$, we distinguish two scenarios: the case without any extra space, i.e., $\gamma = 1$, and the case $\gamma = 1 + \varepsilon$ a constant $\varepsilon > 0$. In the case $\gamma = 1$, we can provide an asymptotically tight analysis.

Theorem 3. *There exists an online algorithm for* ONLINE-SORTING$[1, n]$ *with competitive ratio at most* $18\sqrt{n}$*. Every online algorithm of* ONLINE-SORTING$[1, n]$ *has competitive ratio at least* $\sqrt{n/2}$*.*

As we describe in Section 1.3, this can be seen as an asymptotically tight analysis of an online version of the travelling salesperson problem (TSP) on the real line. Indeed, we can imagine that we must visit n cities on $[0, 1]$ at time steps $1, \ldots, n$. The position of each city is revealed to us in an online fashion and we immediately have to decide the time step where we visit this city. In addition to packing and TSP, we believe that the online sorting problem can be useful when studying other online problems as well.

In contrast to Theorem 3, when the available space is a constant factor larger than n, there exists an algorithm with competitive ration $n^{o(1)}$.

Theorem 4. *For any* $\varepsilon > 0$*, there exists an algorithm for* ONLINE-SORTING$[1 + \varepsilon, n]$ *with competitive ratio* $2^{O(\sqrt{\log n \log \log n})}$*.*

We note that there is an exponential gap between the lower and upper bounds in Theorem 1 and Theorem 4. It is an interesting open problem to close this gap, say for ONLINE-SORTING$[2, n]$.

There is a trivial n-competitive algorithm STRIP-PACKING that places each of the n pieces as deep into the strip as possible. Improving upon this turns out to be quite challenging. We present an online algorithm with competitive ratio $O(n^{\log 3 - 1} \log n) = O(n^{0.59})$, where $\log x$ denotes the base-2 logarithm of x.

Theorem 5. *There exists an algorithm for* STRIP-PACKING *with competitive ratio* $O(n^{\log 3 - 1} \log n)$*, where* n *is the number of pieces.*

Another interesting open problem is to improve upon this. Is it for example possible to obtain an $n^{o(1)}$-competitive algorithm for STRIP-PACKING as we have for ONLINE-SORTING$[1 + \varepsilon, n]$? Because the sorting problem is much simpler than the packing problem, the lower bound from Theorem 1 implies a lower bound for STRIP-PACKING, but the algorithm behind Theorem 4 does not lead to any packing algorithm.

1.1 The necessity of sorting pieces by slope

Our results in Theorem 2 are in contrast to translational offline packing of convex polygons for which constant factor approximations exist. In a recent paper, Alt, de Berg, and Knauer [5, 6] gave a constant-factor approximation algorithm for offline translational packing of convex polygons so as to minimize the area of their bounding box. The algorithm works by first grouping the pieces into exponentially increasing height classes and then sorting the pieces in each height class by the slopes of their *spine segments*; see Figure 1. The spine segment of a piece is the line segment from the bottommost to the topmost corner. Placing the pieces in rows in this sorted order (so that each row appears as a fan-like pattern) yields a compact packing with constant density.

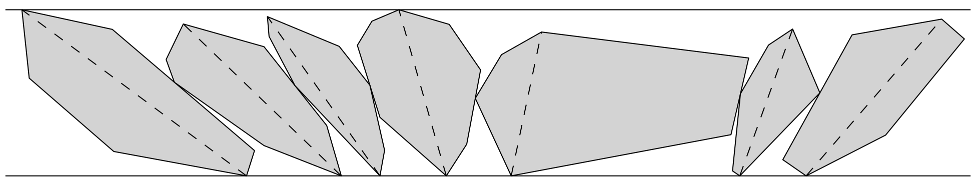


Figure 1: A fan-like packing of pieces of nearly equal height. The pieces are sorted according to the slope of their spine segments (dashed).

We show that similar procedures yield constant factor approximations for the offline version of strip packing, bin packing, square packing and perimeter packing.

Theorem 6. *There are polynomial-time offline algorithms for the following packing problems:*

(a) Offline-Strip-Packing*, 32.7-approximation algorithm.*

(b) Offline-Bin-Packing*, 10.1-approximation algorithm if the diameter of all pieces is bounded by* $1/10$*.*

(c) Offline-Perimeter-Packing*, 8.9-approximation algorithm.*

(d) Offline-Square-Packing[1/10]*, in particular, every set of convex polygons of diameter and total area at most* $1/10$ *can be packed into the unit square.*

The contrast between Theorem 2 and Theorem 6 shows that sorting the pieces by the slope of their spine segments is essential for obtaining an efficient packing. In particular, we use the lower bound from Theorem 1 to create an adaptive stream of pieces that will force any packing algorithm to use excessive space. In the reduction, the numbers to be sorted in the online sorting problem correspond to the slopes of the spine segments in the packing problems, and the impossibility of placing the numbers in nearly sorted order implies that the packing algorithm is forced to produce an arrangement that is far from an optimal fan-like pattern.

1.2 Relation to previous work on packing

The literature on online packing problems is vast. See the surveys of Christensen, Khan, Pokutta, and Tetali [13], Epstein and van Stee [17], van Stee [41, 42], and Csirik and Woeginger [15] for an overview. Below we survey the most important results for each of the types of packing problems mentioned in Theorem 2. Let us highlight that when the pieces are restricted to axis-parallel rectangles, there are online algorithms with constant competitive ratios solving all the problems of Theorem 2.

Strip packing. In strip packing, we have a horizontal strip of height 1 bounded by a vertical segment to the left but unbounded to the right. The goal is to place the pieces in the strip so as to minimize the length of the part of the strip that has been used. Milenkovich [34] and Milenkovich and Daniels [35] described exact algorithms for offline strip packing where the pieces are simple or convex polygons.

Baker and Schwarz [9] described the first algorithms for online strip packing of rectangular pieces. The FFS (First Fit Shelf) algorithm has a competitive ratio of 6.99 under the assumption that the height of each rectangle is at most 1. Ye, Han, and Zang [43] improved the algorithm and obtained a competitive ratio of $7/2 + \sqrt{10} \approx 6.6623$ without the restriction on rectangle heights. Restricting the attention to large instances, FFS has an asymptotic competitive ratio that can be made arbitrarily close to 1.7. Csirik and Woeginger [16] described an improved algorithm with an asymptotic competitive ratio arbitrarily close to $h_\infty \approx 1.69103$. This was later improved to 1.58889 by Han, Iwama, Ye, and Zhang [24]. In contrast, we show that when the pieces are convex polygons, then no competitive algorithm exists (Theorem 2 (a)).

Bin packing. In bin packing, we have an unbounded supply of identical containers, and the goal is to pack the pieces into as few containers as possible. As mentioned, online bin packing problems have been studied since the early 1970s [20]. Many papers have been devoted to the study of online translational bin packing axis-parallel rectangular pieces into unit square bins. In long sequences of papers, the upper bound on the asymptotic competitive ratio for this problem has been decreased from 3.25 to 2.5545 and the lower bound has been increased from 1.6 to 1.907 [23]. In this paper, we show that when packing convex polygons instead of axis-parallel rectangles, there is no competitive algorithm (Theorem 2 (b)).

Perimeter packing. In some packing problems, the "container" has no predefined boundaries (contrary to the cases of strip and bin packing and the study of critical densities), but the pieces can be placed anywhere in the plane and the container is dynamically updated as the bounding box or the convex hull of the pieces. The goal is then to minimize the size of the container. In 2D versions of this problem, natural measures of size are the area or the perimeter of the container. Many papers have been written about offline versions of these problems [2, 4, 5, 8, 14, 18, 29, 30, 31, 33, 34, 35, 37, 40].

Online versions have received relatively little attention. Fekete and Hoffmann [19] studied online packing axis-parallel squares so as to minimize the area of their bounding square, and gave an 8-competitive algorithm for the problem. Abrahamsen and Beretta [1] gave a 6-competitive algorithm for the same problem and studied the more general case where the pieces are axis-parallel rectangles and we want to minimize the bounding box, with or without rotations by $90°$ allowed. They gave a 3.98-competitive algorithm for minimizing the perimeter and showed that there exists no competitive algorithm for minimizing the area, when the pieces can be arbitrary rectangles.

If the pieces are convex polygons that can be arbitrarily rotated, then the minimum perimeter problem can be reduced to the case of packing axis-parallel rectangles by first rotating each piece so that a diameter of the piece is horizontal. Then the density of the piece in its axis-parallel bounding box is at least $1/2$, and the algorithm for rectangles can be applied to the bounding box. An interesting question that remained open was therefore whether there is a competitive algorithm for minimizing the perimeter when the pieces are convex polygons that can *not* be rotated. We answer this question in the negative (Theorem 2 (c)).

Critical densities and square packing. The study of critical densities dates back at least to the 1930s. In the famous *Scottisch Book* [32], Auerbach, Banach, Mazur and Ulam gave the following theorem (slightly rephrased) and corollary without a proof.

Theorem (Potato Sack Theorem, [32])**.** *If $\{K_n\}_{n=1}^{\infty}$ is a sequence of convex bodies in $\mathbb{R}^3$, each of diameter $\leq \delta$ and the sum of their volumes is $\leq V$, then there exists a cube with sidelength $s = f(\delta, V)$ such that all the given bodies can disjointly be placed into it when rotations are allowed.*

Corollary. *One kilogram of potatoes can be put into a finite sack.*

A simple proof of the theorem, generalized to an arbitrary dimension, was given by Kosiński [27]. It was asked in [32] to determine the function $f(\delta, V)$, which, in modern terms, means to determine the *critical density.* That is, to find the largest value of V such that a sequence of convex bodies of diameters at most δ and total volume V can always be placed in the unit cube. This theorem has sprouted a lot of interest in determining critical densities in various settings. For instance, Moon and Moser [36] proved that any sequence of d-dimensional cubes of total volume $1/2^{d-1}$ can be packed into the unit cube. As two cubes with sidelengths $1/2 + \varepsilon$, for any $\varepsilon > 0$, cannot be packed in the unit cube, this shows that the critical density of packing cubes into a unit cube is $1/2^{d-1}$ for any $d \geq 1$. Alt, Cheong, Park, and Scharf [7] showed that there exist n 2D unit disks embedded in 3D (with different normal vectors) such that whenever they are placed in a non-overlapping way, their bounding box has volume $\Omega(\sqrt{n})$. It follows that when rotations are not allowed, the critical density of packing convex bodies of bounded diameter into a cube is 0, or, in other words, that one kilogram of potatoes cannot always be put into a finite sack by translation. In contrast to this, the critical density of packing convex polygons of bounded diameter into the unit square by translation is positive when the diameter is sufficiently small, as we prove in Theorem 6 (d).

The study of critical densities likewise makes sense when the pieces appear in an online fashion. A lower bound on the critical density of online packing squares into the unit square has been improved in a sequence of papers [11, 19, 25, 26] from $5/16$ [26] to $2/5$ [11]. Interestingly, Januszewski and Lassak [26] proved that in dimension $d \geq 5$, the critical density of online packing cubes into the unit cube is $1/2^{d-1}$, just as in the offline case.

Lassak and Zhang [28] proved that the Potato Sack Theorem also holds for any dimension $d \geq 1$ when the convex bodies appear online, if rotations are allowed. In order to achieve this, each convex body of volume V is rotated so that it has an axis-parallel bounding box of volume at most $d! \cdot V$. The problem is therefore reduced to online packing axis-parallel boxes. In simplified terms, it is then proved that for some constant $\delta = \delta(d) > 0$, any sequence of axis-parallel boxes of diameter and total area at most δ can be packed online in the d-dimensional unit hypercube. Determining whether the critical density of translational *and* online packing convex 2D polygons is positive remained an interesting question: On one hand, this packing problem is harder than the 2D offline version which has positive critical density (Theorem 6 (d)), and on the other hand, it is easier than the 3D online version which has 0 critical density (since also the 3D offline version has 0 critical density [7]). In this paper, we prove that the 2D *online* version also has critical density 0 (Theorem 2 (d)).

1.3 Online sorting, TSP and scheduling

Theorem 3 can be seen as an asymptotically tight analysis of the traveling salesperson problem (TSP) on the real line, following the *online-list* paradigm: We want to visit n cities in the unit interval $[0, 1]$ over the course of n days, one city per day. The positions of the cities are revealed sequentially to us in an online fashion, and for each city, we have to immediately decide which day to visit that city. Our goal is to minimize the total distance travelled. In fact, we could equally well imagine that we had γn days for our tour which corresponds to Online-Sorting$[\gamma, n]$.

The usually studied version of online TSP follows the *online-time* paradigm. Here, a *server* starts at point 0 at time 0 and moves with at most unit speed. Each request $\sigma_i = (t_i, r_i)$ is revealed at some time t_i and should be visited by the server at time r_i or later. We want to minimize the time before the server has visited all requests $\sigma_1, \ldots, \sigma_n$. This problem has been intensely studied [10]. The distinction between these two paradigms is common in the area of scheduling, but the online-list variant of TSP has apparently not received any attention so far.

Fiat and Woeginger [21] studied a scheduling problem following the online-list paradigm that seems particularly related to online sorting: The goal is to minimize the average job completion time in a system with n jobs and a single machine. In every step, a single new job arrives and must be scheduled to its time slot immediately and irrevocably and without knowledge of the jobs that arrive in later steps. The offline optimum is to schedule the jobs according to their processing times in sorted order. It was shown that no algorithm can be $\log n$-competitive, but that there is a $O(\log^{1+\varepsilon} n)$-competitive algorithm for all $\varepsilon > 0$. For surveys on (online) scheduling, see [3, 12, 22, 38, 39].

1.4 Structure of the paper

In Section 2, we introduce our terminology and notation. In Section 3, we describe the connection between the problems of online sorting and online strip packing. In Section 4, we analyze the online sorting problem and prove Theorems 1, 3 and 4. In Section 5, we study online packing problems and present proofs for Theorems 2 and 5. Finally, in Section 6, we consider the offline versions of the packing problems and prove Theorem 6. See Figure 2 for an overview of the reductions we make.

$$\begin{tikzcd}
 & & \boxed{\text{Strip-Packing, } \varnothing \le \delta} & \\
\boxed{\text{Online-Sorting}[C, n]} \arrow[r] & \boxed{\text{Strip-Packing}} \arrow[ur] \arrow[r] \arrow[dr] & \boxed{\text{Bin-Packing, } \varnothing \le \delta} & \\
 & & \boxed{\text{Square-Packing}[\delta]} \arrow[r] & \boxed{\text{Perimeter-Packing}}
\end{tikzcd}$$

Figure 2: An overview of our reductions. Note that an arrow from problem A to problem B means that an algorithm for B implies an algorithm for A. Here, $\varnothing \le \delta$ means that the diameter of each piece is at most an arbitrary constant $\delta > 0$.

2 Terminology

In the online problems studied in this paper, the input is a stream $\sigma_1, \ldots, \sigma_n$ of objects, and we need to handle each object σ_i before we know the next ones $\sigma_{i+1}, \ldots, \sigma_n$. Here, objects are either real numbers from the unit interval $[0, 1]$, in which case we call them *reals*, or they are convex polygons, in which case we call them *pieces*. The problems will be defined in more detail in Section 3.

Let us now revisit the standard terminology of competitive analysis for an online algorithm $\mathcal{A}$ of a minimization problem $\mathcal{P}$. For any instance I of $\mathcal{P}$, let $\mathrm{OPT}(I)$ denote the cost of the offline optimum solution and $\mathcal{A}(I)$ denote the cost of the solution that $\mathcal{A}$ produces on input I. Let f be a function from the set of instances to the real numbers (the functions f we consider will in fact only depend on n, the number of pieces in I). We say that $\mathcal{A}$ has *(absolute) competitive ratio* $f(I)$ if for all instances I it holds that

$$\mathcal{A}(I) \le f(I) \cdot \mathrm{OPT}(I).$$

If $\mathcal{A}$ has competitive ratio $f(I) \leq c$ for some constant c, then we say that $\mathcal{A}$ is *competitive*.

Similarly, we say that $\mathcal{A}$ has *asymptotic competitive ratio* $f(I)$ if there exists $\beta > 0$ such that for all instances I it holds that

$$\mathcal{A}(I) \leq f(I) \cdot \mathrm{OPT}(I) + \beta.$$

For a point $p \in \mathbb{R}^2$, we denote by $x(p)$ and $y(p)$ the x- and y-coordinates of p, respectively. For a compact set of points $S \subset \mathbb{R}^2$, we define the *diameter* of S as $\max_{p,q \in S} |p - q|$. We furthermore define the *width* and *height* of S as

$$\mathrm{width}(S) := \max_{p,q \in S} |x(p) - x(q)| \quad \text{and} \quad \mathrm{height}(S) := \max_{p,q \in S} |y(p) - y(q)|.$$

If S is a polygon, we denote the area of S as area(S).

A parallelogram P is *horizontal* if P has a pair of horizontal edges. The horizontal edges of P are then called the *base* edges. The *shear* of a horizontal parallelogram P is $x(t) - x(b)$, where t and b are the top and bottom endpoints of a non-horizontal edge of P, respectively.

3 The connection between online sorting and packing

In (translational) Strip-Packing, we have a horizontal strip S of height 1 which is bounded to the left by a vertical segment and unbounded to the right; see Figure 3. We have to pack a stream of convex polygonal pieces appearing online. We must place each piece before we know the next. Specifically, each piece must be placed in S by a translation such that it is interior disjoint from all previously placed pieces. The *occupied* part of S is the part from the left end of S to the vertical line through the rightmost corner of a piece placed in S. The *width* or the *cost* of a packing is the width of the part of S that is occupied, i.e., the horizontal distance from the left end of S to the rightmost corner of a piece placed in S.

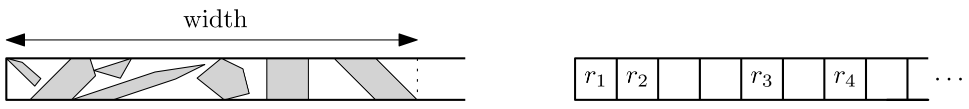


Figure 3: Strip packing and online sorting.

In the problem Online-Sorting$[\gamma, n]$, we are given an array A with γn cells. Each arriving real is contained in the unit interval $[0, 1]$. After all n reals have been placed, define $\mathbf{r} := (r_1, \ldots, r_n)$ to be the numbers according to their left-to-right order in A. Furthermore, define the sentinel values $r_0 := 0$ and $r_{n+1} := 1$. Then the cost is given by

$$\mathrm{cost}(\mathbf{r}) := \sum_{i=0}^{n} |r_{i+1} - r_i|.$$

The offline optimum is achieved when the reals are in sorted order and is then exactly 1.

Figure 2 gives an overview of reductions we make. Arguably, the crucial reduction is that from Online-Sorting to Strip-Packing, as described by the following lemma.

Lemma 7. *If there exists a C-competitive algorithm for* Strip-Packing*, then there exists a $4C$-competitive algorithm for* Online-Sorting$[2C, n]$.

Proof. Suppose that we have a C-competitive algorithm $\mathcal{A}_1$ for Strip-Packing. Let $s_1, \ldots, s_n$, $s_i \in [0, 1]$, be a stream of reals that we wish to sort online in an array A of size $2Cn$. For each real s_i, we construct a parallelogram P_i with height 1, base edges of length $1/n$, and shear s_i. We then present $\mathcal{A}_1$ with P_i and observe where the bottom left corner of P_i is placed in the strip. Let x_i be the x-coordinate of this corner (suppose that the line $x = 0$ forms the left boundary of the strip). We then place s_i in the cell of index $\lfloor nx_i \rfloor$ in the array A, and denote the resulting algorithm $\mathcal{A}_2$. Since the base segments of the parallelograms have length $1/n$, this will not cause any collisions in A.

By sorting the pieces $P_1, \ldots, P_n$ in order of increasing shear and placing them in this order in the strip, we obtain a packing of width at most 2, so $\mathcal{A}_1$ will place all pieces within a prefix of size $2C \times 1$ of the strip. Hence, $\mathcal{A}_2$ will place each real in A.

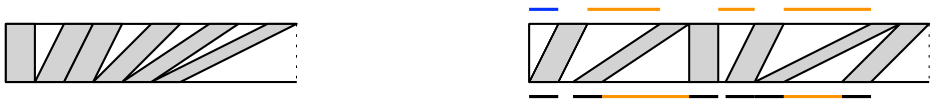


Figure 4: Two strip packings of the same set of parallelograms. Left: The packing we use as an upper bound for the optimum (which is in fact the optimum). Right: An arbitrary packing with the gaps and segments that we count shown as segments outside the strip.

Let $\mathbf{r} := (r_1, r_2, \ldots, r_n)$ be the numbers in the resulting left-to-right order in A produced by $\mathcal{A}_2$, and let P'_i be the parallelogram corresponding to r_i. We have the following contributions to the width of the resulting packing; see also Figure 4.

- Between the vertical left boundary of the strip and the top edge of P'_1, there is a gap of length at least $r_1 = |r_1 - r_0|$.
- If $r_i \leq r_{i+1}$, then there is a gap between the top edges of P'_i and P'_{i+1} of length at least $|r_{i+1} - r_i|$.
- If $r_i > r_{i+1}$, then there is a gap of length at least $|r_{i+1} - r_i|$ between the bottom edges of P'_i and P'_{i+1}.
- The bottom base edges of the pieces have total length $1 \geq |r_{n+1} - r_n|$.

The sum of all these gaps is at least $\text{cost}(\mathbf{r})$, and as half of the sum appear as distances along the top or the bottom edge bounding the strip, we get that the width of the produced packing is at least $\text{cost}(\mathbf{r})/2$. Now, since $\mathcal{A}_1$ is C-competitive, we get that $\text{cost}(\mathbf{r})/2 \leq 2C$, and hence $\text{cost}(\mathbf{r}) \leq 4C$. □

4 Online sorting

In this section, we present upper and lower bounds for the online sorting problem. As a warm up, we consider in Section 4.1 the case where we have no extra space, i.e., we are given n reals in $[0, 1]$ in an online fashion to be inserted into an array of length n. We prove Theorem 3, which gives an asymptotically tight analysis of the optimal competitive ratio in this case. In Section 4.2, we proceed to prove Theorem 1, thereby obtaining a general lower bound on the competitive ratio for Online-Sorting$[\gamma, n]$. Finally, in Section 4.3, we give an upper bound on the competitive ratio of Online-Sorting$[\gamma, n]$ for $\gamma > 1$.

4.1 Tight analysis of Online-Sorting$[1, n]$

For online sorting n numbers in an array A of size n, we can prove asymptotically tight bounds on the optimal competitive ratio.[1] We restate Theorem 3 below.

Theorem 3. *There exists an online algorithm for* Online-Sorting$[1, n]$ *with competitive ratio at most* $18\sqrt{n}$. *Every online algorithm of* Online-Sorting$[1, n]$ *has competitive ratio at least* $\sqrt{n/2}$.

Proof. We start out by proving the lower bound. Let $N := \lfloor \sqrt{2n} \rfloor$. Consider a fixed but arbitrary online algorithm $\mathcal{A}$. We are taking the role of the adversary and present reals of the form k/N with $k = 0, 1, \ldots, N$. Clearly, if all reals are placed in an increasing fashion, the resulting cost is 1. This is the value $\mathcal{A}$ has to compete against.

For a partially filled array, a real is *expensive* if it does not appear in a cell with an empty adjacent cell. Our strategy is as follows. While there exists one, we present an expensive real. If we are able to present n expensive reals, then any two consecutive entries are distinct and differ by at least $1/N$. Consequently, the cost is at least $\frac{n+1}{N} \geq \sqrt{n/2}$. On the other hand, if we reach a point where no real of the form k/N is expensive, we present 0's until all entries are filled. Then, any real of the form k/N will appear next to a 0 in the array, so the total cost is at least $\sum_{k=0}^{N} k/N = (N+1)/2 > \sqrt{n/2}$.

Next we describe an algorithm $\mathcal{A}$ for Online-Sorting$[1, n]$ with competitive ratio at most $18\sqrt{n}$. Define $N_1 := \lfloor \sqrt{n} \rfloor$ and partition $[0, 1]$ into N_1 intervals, $J_1, \ldots, J_{N_1}$, each of length $1/N_1$. Further define $N_2 := 2N_1$

[1]We thank Shyam Narayanan for improving our initial $O(\sqrt{n} \log n)$ upper bound to the asymptotically tight $O(\sqrt{n})$.

and partition the array A into N_2 subarrays of contiguous cells, $A_1, \dots, A_{N_2}$, each of size either $\lfloor n/N_2 \rfloor$ or $\lceil n/N_2 \rceil$. Whenever $\mathcal{A}$ receives a real $x \in J_i$, it checks whether there exists a j such that A_j is not full and already contains a number from J_i. If so, $\mathcal{A}$ places x in the leftmost empty cell of A_j. Otherwise, $\mathcal{A}$ checks if there exists a j such that A_j is empty. If so, $\mathcal{A}$ places x in the leftmost cell of such an A_j.

Finally, if neither of the two above cases occur, then $\mathcal{A}$ recurses on the subarray formed by the union of the empty cells of A.

Ignoring the sentinel values $r_0 := 0$ and $r_{n+1} := 1$, we prove by induction on n that the total cost never exceeds $T(n) = \alpha\sqrt{n}$, where $\alpha := \frac{5\sqrt{2}}{\sqrt{2}-1} \approx 17.1$. This clearly holds for $n \leq \alpha^2$, as then $T(n) \geq n$ which is a trivial upper bound on the total cost. So let $n > \alpha^2$ and assume inductively that the result holds for arrays of length $n' < n$. Let $B \subset \{1, \dots, n\}$ denote the set of indices i such that $A[i]$ is full when the algorithm recurses for the first time. At this point, at most $N_1 - 1$ of the $N_2 := 2N_1$ subarrays $A_1, \dots, A_{N_2}$ can be not completely filled, and each subarray must contain at least one real. In particular, $|B| \geq n/2$. Let $B^c := \{1, \dots, n\} \setminus B$ and define $D_1 := \{1 \leq i < n \mid \{i, i+1\} \subset B^c\}$, and $D_2 := \{1, \dots, n-1\} \setminus D_1$. Define $T_j := \sum_{i \in D_j} |A[i+1] - A[i]|$ for $j \in \{1, 2\}$. The total cost incurred by the algorithm is then $T_1 + T_2$. By the induction hypothesis, $T_1 \leq T(|B^c|) \leq T(\lfloor n/2 \rfloor)$. Furthermore, we can upper bound T_2 as

$$T_2 \leq \frac{n}{N_1} + \frac{3N_2}{2}.$$

Here the first term comes from consecutive entries internal to a subarray that are both filled before the algorithm recurses for the first time (these have pairwise distance at most $1/N_1$). The second term comes from consecutive entries $A[i], A[i+1]$ where $A[i]$ is in some subarray A_j and $A[i+1]$ is in the next A_{j+1}, or $A[i]$ is full when the algorithm recurses for the first time and $A[i+1]$ becomes full in the recursion (in which case we simply bound $|A[i] - A[i+1]| \leq 1$). The total cost is therefore upper bounded by

$$T_1 + T_2 \leq T(\lfloor n/2 \rfloor) + \frac{n}{N_1} + \frac{3N_2}{2} \leq \alpha\sqrt{\frac{n}{2}} + 5\sqrt{n} = \alpha\sqrt{n}.$$

It is easy to check that including the sentinel values, the total cost incurred by the algorithm never exceeds $18\sqrt{n}$. This completes the proof. □

We remark that the constant 18 in Theorem 3 can be significantly reduced by optimizing over N_1 and N_2 and using less crude bounds. We have abstained from doing so in order to obtain a simple exposition.

Having dealt with the case where we have no extra space, we turn to the setting where the array has length γn for some $\gamma > 1$. This is the setting which is important for our reductions to the online packing problems. In the following two sections, we prove lower and upper bounds, respectively, on how good online sorting algorithms can perform in this case.

4.2 Lower bound for the general case of Online-Sorting

Let us restate our lower bound for the competitive ratio of any algorithm for Online-Sorting$[\gamma, n]$, $\gamma \geq 1$.

Theorem 1. *Suppose that $\gamma, \Delta \geq 1$ are such that* Online-Sorting$[\gamma, n]$ *admits a Δ-competitive algorithm, then $\gamma\Delta = \Omega(\log n / \log\log n)$.*

Before delving into the proof, we first describe the high level idea on how to generate an adversarial stream that incurs a high cost for any given algorithm. Assume for simplicity that γ is a constant, e.g., $\gamma = 2$ and that we want to prove a lower bound of Δ on the total cost for some $\Delta = (\log n)^{\Theta(1)}$. We will start by presenting the algorithm with reals coming from a set S of the form $S = \{i/n_0 \mid 0 \leq i \leq n_0\}$ for some $n_0 \leq n$. At any point, we consider the set of maximal intervals of empty cells of the array, call them $I_1, \dots, I_\ell$. For each real $x \in S$, we define the home $H(x)$ as the union of all such intervals I_j such that placing x in I_j incurs no extra cost, i.e., the first non-empty cell to the left or right of I_j contains the real x. (In fact, we will be a little more generous in the actual proof and allow for some small extra cost). If $|H(x)| < \frac{n}{\Delta n_0}$ for some $x \in S$, we simply present the algorithm with copies of x until one is placed outside $H(x)$ and thus has distance at least $1/n_0$ to its neighbours. This will essentially contribute a total cost of $1/n_0$ to the objective function, i.e., an average cost of Δ/n per presented copy of x. Note that this is the correct average cost for a lower bound of Δ. However, it may well be the case that no such $x \in S$ exists (the average size of $H(x)$ for $x \in S$ is $\approx \gamma n/n_0$ which is much larger). In this case, we *coarsen* the set S to a set $S' \subset S$ consisting of every s'th element of S for some $s = \operatorname{polylog} n$ and only present the algorithm with reals from S' from this point on. Now for most $x \in S$, it holds that $H(x) = O(n/n_0)$ and that the distance from x to any real in S' is $\Omega(s/n_0)$. Intuitively, this means that filling up $H(x)$ with elements from S' has a high cost of $s/n \gg \Delta/n$ per presented real. We prove that this implies that we can point to a 'deserted space' consisting of $\Omega(n/\Delta)$ empty cells, in which the algorithm can only place a negligible number of reals without incurring a large total cost of Δ. Now we continue the process starting with S'. In each coarsening step, we specify a 'deserted space' of size $\Omega(n/\Delta)$ and we can importantly enforce that these spaces be disjoint. As the array has $2n$ cells in total, this coarsening can happen at most $O(\Delta)$ times. To ensure that we can in fact perform this coarsening $\Omega(\Delta)$ times, we must ensure that $s^{\Delta} \ll n$, which in turn implies that $\Delta = O(\log n/\log\log n)$.

Proof of Theorem 1. Let $\mathcal{A}$ denote any online algorithm for ONLINE-SORTING$[\gamma, n]$. We may assume that n is sufficiently large and that $\gamma \le \log n/\log\log n$. We will present the algorithm with an (adaptive) stream that incurs a cost of $\Omega(\log n/\gamma \log\log n)$.

Let $C \in [3,4]$ be minimal such that $s := \log^C n$ is an integer and define $\delta := \frac{\log n}{16C\gamma \log\log n}$. For $i \in \mathbb{N}$, we define

$$S_i := \left\{ k \cdot \frac{s^i}{n} \,\middle|\, 0 \le k \le n/s^i \right\}$$

such that $S_1 \supset S_2 \supset \cdots$. We also let $i^* \in \mathbb{N}$ be maximal with $s^{i^*} \le n$. In other words, we define $i^* := \lfloor \log n/C \log\log n \rfloor$. Our adversarial stream will consist of several phases, where in phase $i \ge 1$, we present $\mathcal{A}$ with reals coming from the set S_i. By the end of each phase, we will mark a certain set of currently empty cells of the array. The marked cells will represent parts of the array where $\mathcal{A}$ can only place a limited number of reals without incurring a high cost. Identifying the array with $[m]$, at any given point in time t, i.e., after t reals in $[0,1]$ have been presented to $\mathcal{A}$, we let $F_t \subset [m]$ denote the full cells, $M_t \subset [m]$ denote the marked cells, and $R_t := [m] \setminus (F_t \cup M_t)$ denote the remaining cells. We remark that F_t and M_t need not be disjoint — even though a cell can only get marked when it's empty, $\mathcal{A}$ might insert a real in that cell at a later point in time.

Suppose that we are at time t and in some phase i. For an empty cell $p \in R_t$, we define $N_t(p) \subset [m]$ to be the set consisting of the first non-empty cell to the left and to the right of p. Thus $|N_t(p)| \le 2$. Furthermore, for $x \in S_i$, we define the *home* of x at time t to be the set

$$H_t(x) := \left\{ p \in R_t \,\middle|\, \exists q \in N_t(p) \text{ such that } |A[q] - x| < \frac{s^i}{2n} \right\}.$$

By construction, a cell of A can be contained in at most two homes. We say that the home of x is *small* at time t if $|H_t(x)| < \frac{s^i}{\delta}$. In this case, we say that x is *expensive* at time t.

The adversarial stream is constructed in the phases (were we stop as soon as the array contains n reals).

Phase 0**:**

Present $\mathcal{A}$ with any real from S_1. Proceed to phase 1.

Phase $i \ge 1$**:**

While there exists an expensive real $x \in S_i$ (at some time t):

– Present $\mathcal{A}$ with copies of x until one is placed in a cell of $R_t \setminus H_t(x)$.

– Leave the set of marked cells unchanged.

If no expensive real exists at the current time t:

– Define the set of *well-sized* reals at phase i as $W_i := \left\{ x \in S_i \,\middle|\, \frac{s^i}{\delta} \le |H_t(x)| \le 4\gamma s^i \right\}$.

– Define the set of *deserted reals* at phase i as $D_i := \left\{ x \in W_i \,\middle|\, \forall y \in S_{i+1} \text{ it holds that } |y - x| \ge \frac{s^{i+1}}{12n} \right\}$.

– Define the *deserted space* at phase i as $\mathcal{D}_i := \bigcup_{x \in D_i} H_i(x)$.

– Mark all the cells in $\mathcal{D}_i$ (so that $M_t := M_{t-1} \cup \mathcal{D}_i$).

Note that $s^{i^*+2}/\delta \ge sn/\delta > \gamma n$, so in phase $i^* + 2$, the real 0 is always expensive and the process will therefore eventually stop.

We prove three lemmas below which combine to give our desired result.

Lemma 8. *Let $i \le i^* - 2$. If the algorithm does not terminate in phase i, then $|\mathcal{D}_i| \ge \frac{n}{8\delta}$.*

Proof. We consider the situation in the end of a phase i where the algorithm has not terminated. Since for each $x \in S_i \setminus W_i$, we have $H_t(x) > 4\gamma s^i$ and a cell can lie in at most two homes, it follows that $|S_i \setminus W_i| \leq 2 \cdot \frac{\gamma n}{4\gamma s^i} \leq |S_i|/2$, and so $|W_i| \geq |S_i|/2$. Now S_{i+1} can be obtained from S_i by including every s'th elements from S_i in increasing order starting with 0. If we define $D'_i := \left\{x \in S_i \mid \forall z \in S_{i+1} \text{ it holds that } |z-x| \geq \frac{s^{i+1}}{12n}\right\}$, it then follows that $|D'_i| \geq \frac{3}{4}|S_i|$. Here we used the assumption that $i \leq i^* - 2$ which implies that that S_i consists of sufficiently many reals (at least s^2) for the bound to hold. It follows that $|D_i| = |W_i \cap D'_i| \geq |S_i|/4$. Again using that each cell is contained in at most two homes, we have that

$$|\mathcal{D}_i| \geq \frac{1}{2} \cdot \frac{|S_i|}{4} \cdot \frac{s^i}{\delta} \geq \frac{n}{8\delta}. \qquad \square$$

Lemma 9. *Let $\alpha > 0$ and assume that the algorithm does not terminate in phase i. If at least $56\alpha n\gamma/s$ reals from S_{i+1} are placed in $\mathcal{D}_i$, then the total cost is at least α.*

Proof. Write $\mathcal{D}_i = \bigcup_\ell J_\ell$ as a disjoint union of maximal intervals of empty cells in $\mathcal{D}_i$. For each ℓ, we have that $|J_\ell| \leq 4\gamma s^i$. If at least $56\alpha n\gamma/s$ reals are placed in $\mathcal{D}_i$, it follows that at least $14\alpha n/s^{i+1}$ of the intervals receive at least one real from S_{i+1}. For such an interval J_ℓ, we have by the definition of a home, that one of the (up to two) non-empty cells immediately to the left and right of J_ℓ contains a real x of distance at most $\frac{s^i}{2n}$ to an element of D_i. However, any $x' \in S_{i+1}$ placed in J_ℓ has distance at least $\frac{s^{i+1}}{12n}$ to any element of D_i. It follows that $|x - x'| \geq \frac{s^i}{2n}(\frac{s}{6} - 1) \geq \frac{s^i}{2n}\frac{s}{7} = \frac{s^{i+1}}{14n}$, assuming that n and hence s are sufficiently large. Since the J_ℓ are disjoint, it easily follows that the total cost is at least $\frac{s^{i+1}}{14n} \cdot \frac{14\alpha n}{s^{i+1}} = \alpha$. $\square$

Lemma 10. *If $\mathcal{A}$ assigns n_0 reals to unmarked cells, during phases $1, \dots, i^*$, then the total cost is at least $\frac{\delta n_0}{4n} - 1$.*

Proof. Write a_i for the number of reals that the algorithm assigns to unmarked cells during phase i, so that $\sum_{i \leq i^*} a_i = n_0$. Let $b_i := \frac{a_i}{\lceil s^i/\delta \rceil}$ and $c_i := \lfloor b_i \rfloor$. We make the following claim.

Claim. *The total cost is at least $\sum_{i \leq i^*} c_i \cdot \frac{s^i}{2n}$.*

Proof of Claim. Note that during the while-loop of our algorithm, the assumption that the home of x is small implies that it must happen at least c_i times during phase i that a real from S_i is placed outside its home. When an real x gets placed in a cell $p \in R_t \setminus H_t(x)$ outside its home, the reals stored in the cells in $N_t(p)$ have distance at least $\frac{s^i}{2n}$ to x. We say that an interval of cells $[p_1, p_2]$ form an *i-jump* if $|A[p_1] - A[p_2]| \geq \frac{s^i}{2n}$. We say that an i-jump $[p_1, p_2]$ and a j-jump $[p'_1, p'_2]$ are disjoint if $(p_1, p_2) \cap (p'_1, p'_2) = \emptyset$. We will show that we can find a collection $\mathcal{J}$ of such jumps such that (1) the jumps in $\mathcal{J}$ are pairwise disjoint and (2) we can make a partition $\mathcal{J} = \bigcup_{i \geq 1} \mathcal{J}^{(i)}$ such that $\mathcal{J}^{(i)}$ contains at least c_i i-jumps. The claim then follows immediately by the triangle inequality.

To show the existence of such a collection $\mathcal{J}$, suppose we at step t in some phase i are given a collection of ($\leq i$)-jumps $\mathcal{J}_t$ and a partition $\mathcal{J}_t = \bigcup_{j \leq i} \mathcal{J}_t^{(j)}$ such that $\mathcal{J}_t^{(j)}$ consists of j-jumps. Suppose further that we place a real x in a cell $p \in R_t \setminus H_t(x)$ at step t. Since x was placed outside of its home, any of the (up to two) intervals with one endpoint at p and the other at a cell of $N_t(p)$ will form an i-jump. Now if p is not contained in a jump of $\mathcal{J}_t$, we can easily extend to a collection $\mathcal{J}_{t+1} = \bigcup_{j \leq i} \mathcal{J}_{t+1}^{(j)}$ which still satisfies (1) and where $\mathcal{J}_{t+1}^{(i)}$ contains one further i-jump. We simply add the i-jump between p and either of its neighbours in $N_t(p)$. Suppose on the other hand that p is contained in a j-jump $[q_1, q_2]$ of $\mathcal{J}_t^{(j)}$ for some $j \leq i$. In this case, we in particular have that $|N_t(p)| = 2$ and we write $N_t(p) = \{p_1, p_2\}$ where $p_1 < p_2$. Then $[p_1, p]$ and $[p, p_2]$ are both i-jumps and in particular j-jumps for $j \leq i$. We then replace $[q_1, q_2]$ with $[p_1, p]$ in $\mathcal{J}_{t+1}^j$ and include $[p, p_2]$ in $\mathcal{J}_{t+1}^i$ to obtain a collection $\mathcal{J}_{t+1} = \bigcup_{j \leq i} \mathcal{J}_{t+1}^{(j)}$ having the same number of j-jumps for $j < i$ but one extra i jump compared to $\mathcal{J}_t$. The existence of the collection $\mathcal{J}$ follows immediately from these observations. $\square$

Now assuming that n is sufficiently large, we have that $b_i \geq \frac{\delta a_i}{2s^i}$. Moreover, using the definition of i^*, it is easy to check that $\sum_{i \leq i^*} s^i \leq 2n$. Combining this with the claim, we can lower bound the total cost by

$$\sum_{i \leq i^*} c_i \cdot \frac{s^i}{2n} \geq \sum_{i \leq i^*} (b_i - 1) \cdot \frac{s^i}{2n} \geq \sum_{i \leq i^*} \frac{\delta a_i}{4n} - \sum_{i \leq i^*} \frac{s^i}{2n} \geq \frac{\delta n_0}{4n} - 1,$$

as desired. ☐

We now combine the two lemmas to prove a lower bound on the cost incurred by the algorithm on our adversarial stream. Recall that $C = 3$ and $\delta = \frac{\log n}{16C\gamma \log\log n}$. Suppose for contradiction that the algorithm has not terminated by the end of phase $i^* - 2$. Using Lemma 8 and the fact that the sets $(\mathcal{D}_j)_{j \leq i^*-2}$ are disjoint, we obtain that

$$\left| \bigcup_{j < i^*} \mathcal{D}_j \right| \geq (i^* - 2)\frac{n}{8\delta} > \frac{\log n}{2C \log\log n} \frac{n}{8\delta} \geq \gamma n,$$

assuming that n is sufficiently large. This is a contradiction as there are only γn cells in A. Thus, the algorithm must terminate during some phase $i_0 \leq i^* - 2$. Now the algorithm must either place at least $n/2$ reals in marked cells or $n/2$ reals in unmarked cells. In the former case, there must be a phase i in which $\frac{n}{2i^*}$ reals a placed in $\mathcal{D}_i$, and hence it follows from Lemma 9 that the total cost is $\Omega(\frac{s}{i^* \gamma}) = \Omega(\log n)$. In the latter case, it follows from Lemma 10 that the total cost is at least $\frac{\delta}{8} - 1 = \Omega\left(\frac{\log n}{\gamma \log\log n}\right)$. ☐

4.3 Upper bound for the general case of Online-Sorting

In this section, we design an algorithm that shows the following theorem.

Theorem 4. *For any $\varepsilon > 0$, there exists an algorithm for* Online-Sorting$[1+\varepsilon, n]$ *with competitive ratio* $2^{O(\sqrt{\log n \log\log n})}$.

First, we prove a slightly more general lemma, and then we instantiate it with the right set of parameters to obtain Theorem 4.

Lemma 11. *Let $\delta \in (0, 1/2)$ and $[\alpha, \alpha + \beta) \subseteq [0, 1]$. Then, for any $k \geq 1$ with $k \leq 1/(2\delta) + 1$ there exists an algorithm that solves* Online-Sorting $[1 + 2k\delta, n]$ *over any stream of reals $r_1, \ldots, r_n \in [\alpha, \alpha + \beta)$ achieving a cost of $\beta \cdot n^{1/(k+1)} \delta^{-O(k+1)}$.*

Proof. We prove the statement by induction on k. The base case of $k = 1$ follows directly from Theorem 3: in fact, it is sufficient to apply the mapping $x \mapsto \alpha + \beta x$ to the reals in our stream and notice that the resulting cost also shrinks by a factor β. We call this version of the algorithm $\texttt{OnlineSorter}_1$.

For the induction step, we define the algorithm $\texttt{OnlineSorter}_k$ using $\texttt{OnlineSorter}_{k-1}$ as a subroutine. Let $b := \lfloor n^{1/(k+1)} \rfloor$, $n' := \delta n^{k/(k+1)}$ and $w := (1 + 2(k-1)\delta) \cdot n'$. For $i \geq 1$, we define the box

$$B_i := [(i-1) \cdot w, i \cdot w) \,.$$

We define the pointer vector $p : [b] \longrightarrow \mathbb{N}$ and initialize $p(i) = i$ for each $i \in [b]$. We define the set of active boxes $\mathcal{A} := \{B_{p(i)} \,|\, i \in [b]\}$ and let $\mathcal{R} := \max_{i \in [b]} p(i) \cdot w$ denote the rightmost cell index in any active box. We say $B_{p(i)}$ is *full* if it contains n' reals. Given a real $x \in [\alpha, \alpha + \beta)$, let $i \in [b]$ be such that $(i-1) \cdot \beta \leq (x - \alpha)b < i \cdot \beta$. If $B_{p(i)}$ is not full, we place x into $B_{p(i)}$ using $\texttt{OnlineSorter}_{k-1}$. Otherwise, we assign a new active box and set $p(i) := \max_{j \in [b]} p(j) + 1$, update $\mathcal{A}$ and $\mathcal{R}$ accordingly, and then place x into $B_{p(i)}$ using $\texttt{OnlineSorter}_{k-1}$.

By the inductive hypothesis, $\texttt{OnlineSorter}_{k-1}$ can place n' reals into an array of size w. Hence, to prove correctness we only have to ensure that the set of used boxes are contained in the array, i.e., that $\mathcal{R} \leq (1 + 2k\delta)n$. We show that the total number of empty cells in $[1, \mathcal{R}]$ is at most $2k\delta n$ and therefore there must be n full cells in $[1, (1 + 2k\delta)n]$.

Let $\{B_1, \ldots, B_\ell\}$ be the set of boxes we used (either full or active), so that $[1, \mathcal{R}] = \bigcup_{i=1}^{\ell} B_i$. We partition $[\ell]$ into F and E so that $F := \{i \in [\ell] \,|\, B_i \text{ is full}\}$ and $E := [\ell] \setminus F$. Denote with $\mathcal{E}(B_i)$ the number of empty cells in box B_i. If $i \in F$, we have $\mathcal{E}(B_i) \leq w - n' = 2(k-1)\delta \cdot n'$. Moreover, $|F| \leq {}^{n}\!/_{n'}$ and therefore

$$\sum_{i \in F} \mathcal{E}(B_i) \leq \frac{n}{n'} \cdot 2(k-1)\delta \cdot n' = 2(k-1)\delta n.$$

If $i \in E$, we use the trivial bound $\mathcal{E}(B_i) \leq w$, moreover $j \in E$ implies $B_j \in \mathcal{A}$, hence $|E| \leq b$. This yields

$$\sum_{i \in E} \mathcal{E}(B_i) \leq wb \leq (1 + 2(k-1)\delta)n' \cdot n^{1/(k+1)} = (1 + 2(k-1)\delta) \cdot \delta n.$$

Putting everything together, we obtain

$$\sum_{i \in [\ell]} \mathcal{E}(B_i) \leq 2(k-1) \cdot \delta n + (1 + 2(k-1)\delta) \cdot \delta n \leq 2k\delta n,$$

where the last inequality holds because $2(k-1)\delta \leq 1$ by assumption on k.

We are left to bound the total cost for which we likewise use induction. By $\text{cost}^{(k)}(r_1, \ldots, r_n)$, we denote the cost incurred by algorithm $\texttt{OnlineSorter}_k$ when facing the stream of reals $r_1, \ldots, r_n$. We prove that there exists $C > 0$ such that $\text{cost}^{(k)}(r_1, \ldots, r_n) \leq \beta \cdot n^{1/(k+1)} \delta^{-C \cdot (k+1)}$ for any stream of reals $r_1, \ldots, r_n$ with $r_i \in [\alpha, \alpha + \beta)$. For $k = 1$, we already explained above how Theorem 3 implies that $\texttt{OnlineSorter}_1$ places a stream $r_1, \ldots, r_n \in [\alpha, \alpha + \beta)$ with a cost of $\text{cost}^{(1)}(r_1, \ldots, r_n) = 18\beta\sqrt{n}$, and we can choose C accordingly.

Now suppose $k > 1$, and let $\{B_1, \ldots, B_\ell\}$ be the set of full or active boxes. We have $\ell \leq \mathcal{R}/w \leq (1 + 2k\delta)n/w \leq 3n^{1/(k+1)}\delta^{-1}$. We can think of our algorithm as partitioning the stream $r_1, \ldots, r_n$ into substreams according to the box in which each real is placed. For $i \in [\ell]$, denote the substream of length L_i placed in B_i with $y^i_1, \ldots, y^i_{L_i}$ so that we have $\{r_1, \ldots, r_n\} = \bigcup_{i=1}^{\ell} \{y^i_1, \ldots, y^i_{L_i}\}$. Moreover, $L_i \leq n'$ for each $i \in [\ell]$. Define $\alpha' := (i-1) \cdot \beta n^{-1/(k+1)}$ and $\beta' := \beta n^{-1/(k+1)}$, then for each $j \in [L_i]$ it holds $\alpha' \leq y^i_j < \alpha' + \beta'$. By the induction hypothesis, the cost induced by the recursive call of $\texttt{OnlineSorter}_{k-1}$ on box B_i is bounded by

$$\begin{aligned}
\text{cost}^{(k-1)}\left(y^i_1, \ldots, y^i_{L_i}\right) &\leq \beta' \cdot \delta^{-Ck} \left(L_i\right)^{1/k} \\
&\leq \beta n^{-1/(k+1)} \cdot \delta^{-Ck} \left(\delta n^{k/(k+1)}\right)^{1/k} \\
&\leq \beta \cdot \delta^{-Ck}.
\end{aligned}$$

Now we are ready to estimate the total cost of $\texttt{OnlineSorter}_k$ as the sum of costs generated inside a box B_i or between any two consecutive boxes:

$$\begin{aligned}
\text{cost}^{(k)}\left(r_1 \ldots r_n\right) &\leq \ell\beta + \sum_{i=1}^{\ell} \text{cost}^{(k-1)}\left(y^i_1, \ldots, y^i_{L_i}\right) \\
&\leq \ell\beta \cdot \left(1 + \delta^{-Ck}\right) \\
&\leq 3n^{1/(k+1)}\delta^{-1}\beta \cdot \left(1 + \delta^{-Ck}\right) \\
&\leq \beta \cdot n^{1/(k+1)}\delta^{-C \cdot (k+1)}
\end{aligned}$$

where the last inequality holds if we choose C large enough. □

Finally, we can prove Theorem 4.

Proof of Theorem 4. We apply Lemma 11 choosing $\alpha := 0$, $\beta := 1$, $k := \sqrt{\log n / \log\log n}$ and $\delta := \varepsilon/(2k)$. It uses $(1 + 2k\delta)n \leq (1 + \varepsilon)n$ memory cells and yields a cost of

$$n^{1/(k+1)} \cdot \delta^{-O(k+1)} = 2^{O(\log n/k + k\log(2k/\varepsilon))} = 2^{O(\sqrt{\log n \log\log n})}.$$ □

5 Online packing

In this section, we consider various online packing problems. In Section 5.1, we show that Strip-Packing does not allow for a competitive online algorithm. In fact, the argument generalizes to the other important online packing problems Square-Packing, Perimeter-Packing and Bin-Packing. This holds even when all pieces have diameter less than an arbitrarily small constant $\delta > 0$.

In Section 5.2, we present an online algorithm for convex strip packing. A naive greedy algorithm for this problem places each new piece as deep into the strip as possible, and this algorithm is n-competitive, where n is the number of pieces. Our algorithm is $O(n^{0.59})$-competitive.

5.1 Lower bounds — no competitive algorithms

Theorem 1 and Lemma 7 yield the following corollary.

Corollary 12. Strip-Packing *does not allow for a competitive online algorithm, even when the diameter of each piece is at most* 2*. Specifically, for every online algorithm $\mathcal{A}$, there exists a stream of n parallelograms of diameter at most* 2 *such that $\mathcal{A}$ produces a packing of width $\Omega(\sqrt{\log n/\log\log n})$, while the optimal offline packing has width at most* 2.

Proof. Suppose that Strip-Packing has a C-competitive algorithm $\mathcal{A}$. By Lemma 7, we get a $4C$-competitive algorithm $\mathcal{A}_2$ for Online-Sorting$[2C, n]$. Theorem 1 implies that $8C^2 = \Omega(\log n/\log\log n)$. In particular, $C = \Omega(\sqrt{\log n/\log\log n})$.

Specifically, the proof of Theorem 1 yields a stream of n reals where $\mathcal{A}_2$ incurs a cost of $\Omega(\sqrt{\log n/\log\log n})$. By the proof of Lemma 7, this translates to a stream of n parallelograms that can be packed in a strip of width 2, while $\mathcal{A}_1$ produces a packing of width $\Omega(\sqrt{\log n/\log\log n})$. □

We even get a non-constant asymptotic lower bound.

Corollary 13. *For any algorithm $\mathcal{A}$ for* Strip-Packing*, a lower bound on the asymptotic competitive ratio of $\mathcal{A}$ is $\Omega(\sqrt{\log n/\log\log n})$, where n is the number of pieces, even when restricted to pieces of diameter less than* 2.

Proof. Assume for contradiction that $\mathcal{A}$ has asymptotic competitive ratio $g(n) = o(\sqrt{\log n/\log\log n})$. Let $\mathcal{P}(\mathcal{A})$ be the stream described in Corollary 12, where $\mathcal{A}$ produces a packing of width $\Omega(\sqrt{\log n/\log\log n})$, while the optimal packing has width at most 2.

There exists a function f such that: (i) $f(n)\cdot g(n) \le \sqrt{\log n/\log\log n}$, (ii) $f(n) \le n^2$, (iii) $f(n) = \omega(1)$, and (iv) f is non-decreasing. If we append $\sqrt{f(n)}$ unit squares to the stream $\mathcal{P}(\mathcal{A})$, we obtain a stream $\mathcal{Q}(\mathcal{A})$ of $n' := n + \sqrt{f(n)} \le 2n$ pieces. Taking $\mathcal{Q}(\mathcal{A})$ as input, $\mathcal{A}$ produces a packing of width $\Omega(\sqrt{\log n/\log\log n})$ while the optimum is $\Theta(\sqrt{f(n)})$. This implies that $\mathcal{A}$ has a competitive ratio of $\Omega\left(\sqrt{\frac{\log n/\log\log n}{f(n)}}\right) = \omega\left(\frac{\sqrt{\log n'/\log\log n'}}{f(n')}\right)$ on instances having arbitrary large (at least $\sqrt{f(n)}$) optimal offline cost, contradiction. □

The insights of Corollaries 12 and 13 imply negative answers also for various other online packing problems.

Theorem 2. *The following holds, where n is the number of pieces:*

(a) Strip-Packing *does not allow for a competitive online algorithm, even if all pieces have diameter at most δ for any constant $\delta > 0$. In particular, the competitive ratio of any algorithm is $\Omega(\sqrt{\log n/\log\log n})$.*

(b) Bin-Packing *does not allow for a competitive online algorithm, even if all pieces have diameter at most δ for any constant $\delta > 0$. In particular, the competitive ratio of any algorithm is $\Omega(\sqrt{\log n/\log\log n})$.*

(c) Perimeter-Packing *does not allow for a competitive online algorithm, even if all pieces have diameter at most δ for any constant $\delta > 0$. In particular, the competitive ratio of any algorithm is $\Omega(\sqrt[4]{\log n/\log\log n})$.*

(d) SQUARE-PACKING[δ] *does not allow for an online algorithm for any* $\delta \in (0,1]$. *In particular, for any algorithm and infinitely many* n, *there exists a stream of* n *pieces of total area* $O(\sqrt{\log\log n/\log n})$ *that the algorithm cannot pack in the unit square.*

Here, (a) and (b) even hold in the asymptotic sense, i.e., if we restrict ourselves to instances with offline optimal cost at least C, *for any constant* $C > 0$.

Proof. For each of the four packing problems, we consider an arbitrary algorithm $\mathcal{A}^*$, where $\mathcal{A}^*$ packs pieces into a container S^*. Here, S^* may be a strip, a set of bins etc., depending on the problem at hand. The main idea in all the proofs is to *cover* S^* by (rotated) substrips of a strip S of height $1/c$ for a (large) constant c, so that we get a correspondence between points in S^* and S. By feeding $\mathcal{A}^*$ with a stream of parallelograms and observing where they are placed in S^*, we get an online algorithm $\mathcal{A}$ for packing (slightly modified) parallelograms into S. We can then by Corollary 13 choose a stream that forces $\mathcal{A}$ to produce a packing much larger than the optimal one, which implies that $\mathcal{A}^*$ has likewise produced a bad packing. This gives a lower bound on the competitive ratio of $\mathcal{A}^*$.

For a horizontal parallelogram P, a *2-extended* copy is a parallelogram obtained by taking two copies of P and identifying the bottom base segment of one with the top segment of the other copy. We now prove each statement in the theorem.

(a) Suppose that $\mathcal{A}^*$ is an algorithm for STRIP-PACKING restricted to pieces of diameter at most $\delta > 0$. The algorithm $\mathcal{A}^*$ packs pieces into a strip S^* of height 1. We define $c := \lceil 4/\delta \rceil$. Recall that S is a strip of height $1/c$. We cut S into substrips $S_1, S_2, \ldots$, each of which is a rectangles of size $1 \times 1/c$. We rotate these by 90° and use them to cover the strip S^*. Let S_i^* be the part of S^* corresponding to S_i, so that the rectangles $S_1^*, S_2^*, \ldots$ appear in this order from left to right in S^*; see Figure 5.

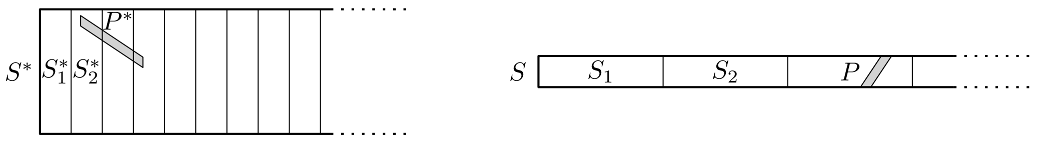


Figure 5: The figure shows the correspondence between the strips S^* and S.

We now define an algorithm $\mathcal{A}$ for strip packing horizontal parallelograms of height $1/c$ and diameter at most $2/c$ into S as follows. Let P be a piece to be packed in S, and let P^* be the piece obtained from rotating the 2-extended copy of P by 90°. Then P^* has diameter at most $4/c = \delta$. We now feed P^* to $\mathcal{A}^*$. We observe where $\mathcal{A}^*$ places P^* in S^*. Since P^* has width $2/c$, it intersects both the left and right vertical edge of a substrip S_i^*. Then, in particular, $P' := P^* \cap S_i^*$ is congruent to P. We now place P in the substrip S_i as specified by the placement of P' in S_i^*. As $\mathcal{A}^*$ does not place pieces in S^* so that they overlap, this approach will produce a valid packing in S.

By Corollary 13, there exists for arbitrarily large n and α a stream I of n pieces of total area α where $\mathrm{OPT}(I) = O(\alpha)$ whereas $\mathcal{A}(I) = \Omega(\sqrt{\log n/\log\log n})$. Let I^* be the stream of 2-extended pieces which we feed to $\mathcal{A}^*$. We then have $\mathrm{OPT}(I^*) = O(\alpha)$, as we can just sort the pieces I^* by the slope of their spine segments and place them in one long row in S^*, which will form a packing of cost at most $\mathrm{OPT}(I) + 2/c = O(\alpha)$. Since $\mathcal{A}(I) \le c \cdot \mathcal{A}^*(I^*)$, we also have $\mathcal{A}^*(I^*) = \Omega(\sqrt{\log n/\log\log n})$. The statement then follows.

(b) The proof is almost identical to that of (a), so we only describe the parts that are different. Suppose that $\mathcal{A}^*$ is an algorithm for BIN-PACKING restricted to pieces of diameter at most $\delta > 0$. Here, $\mathcal{A}^*$ packs pieces into unit square bins $B_1, B_2, \ldots$.

We again partition S into substrips and cover the boxes $B_1, B_2, \ldots$ with these, as shown in Figure 6. We then get a strip packing algorithm $\mathcal{A}$ in S in a similar way as in (a), and obtain a lower bound on the asymptotic competitive ratio of $\mathcal{A}^*$ in a similar way.

(c) Suppose there exists a C-competitive online algorithm $\mathcal{A}^*$ for PERIMETER-PACKING. We prove that this implies the existence of an algorithm for SQUARE-PACKING[$1/160C^2$], and it follows from (d) that

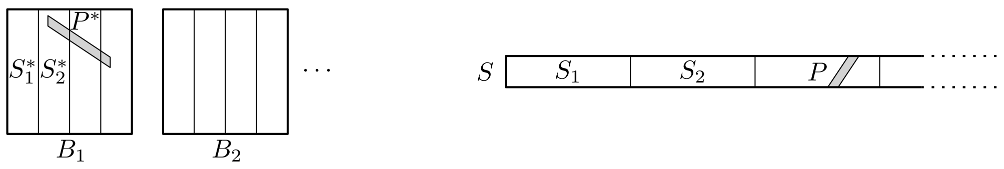


Figure 6: The figure shows the correspondence between the bins $B_1, B_2, \ldots$ and S.

$C = \Omega(\sqrt[4]{\log n / \log\log n})$. By Theorem 6 (c), every set of convex polygons of diameter at most $1/10$ and total area at most $1/10$ can be packed in a 1×1 square. Scaling by $1/4C$, we get that convex polygons of diameter at most $1/40C$ and total area at most $1/160C^2$ can be packed in a $1/4C \times 1/4C$-square. In particular, the same holds if we bound the diameters to be at most $1/160C^2$. Hence, such a set of polygons can be packed in a box of perimeter $1/C$. It follows that $\mathcal{A}^*$ will produce a packing with a convex hull of perimeter at most 1. Therefore, the packing will be contained in the disk of radius $1/2$ centered at any corner of the first piece P_1, and in particular in the unit square centered at this corner. In other words, we have defined an algorithm for the problem Square-Packing$[1/160C^2]$, and the claim follows.

(d) Consider an online algorithm $\mathcal{A}^*$ for packing a stream of pieces of diameter at most $\delta \in (0, 1]$ into the unit square. For arbitrarily large values of n, we prove that there exists a stream of pieces of diameter at most δ and total area $O(\sqrt{\log\log n / \log n})$ that $\mathcal{A}^*$ cannot pack into the unit square. Let $d > 0$ be a lower bound on the multiplicative constant hidden in the first Ω-symbol of Corollary 13. We use the same setup as in (b), just with a single box B, which is covered by the $c := \lfloor \sqrt[4]{d^2 \log n \log\log n} \rfloor$ first substrips of S. By Corollary 13, there exists a stream $P_1, \ldots, P_n$ of pieces such that $\mathcal{A}$ produces a packing of width $\sqrt[4]{d^2 \log n / \log\log n}$, so that $\mathcal{A}^*$ cannot fit the stream of 2-extended pieces in B. We may without loss of generality assume that n is sufficiently large that $4/c \leq \delta$, so that the 2-extended pieces have diameter at most δ. The pieces have total area $O(1/c^2) = O(\sqrt{\log\log n / \log n})$, and the statement follows. □

5.2 A better-than-naive algorithm for strip packing

You are asked to suggest an algorithm for online translational strip packing of convex pieces. What is the first approach that comes to your mind? It may very well be the following greedy algorithm: For each piece P_i that appears, place P_i as far left into the (horizontal) strip as possible. This algorithm is n-competitive: Indeed, it will occupy no more than $n \cdot \max_i \operatorname{width}(P_i)$ of the strip, and the optimum must occupy at least $\max_i \operatorname{width}(P_i)$. This bound is unfortunately also essentially tight: Consider a sequence of very skinny pieces of height 1 and width 1, but with slopes alternating between 1 and -1. The algorithm will produce a packing of width n, while the optimum has width slightly more than 2 (depending on the actual fatness of the pieces).

We found it surprisingly difficult to develop an algorithm provably better than the naive algorithm, but in the following, we will describe an $O(n^{\log 3 - 1} \log n)$-competitive algorithm (note that $\log 3 - 1 < 0.59$). We denote the algorithm `OnlinePacker`.

We first describe the algorithm and carry out the analysis when all the pieces are parallelograms of a restricted type and then show, in multiple steps, how the method generalizes to arbitrary convex pieces. By rescaling, we may without loss of generality assume that the first piece presented to the algorithm has width 1.

5.2.1 Algorithm for horizontal parallelograms of height 1 and width ≤ 1

In the following we describe and analyze the algorithm `OnlinePacker` for online translational strip packing under the assumption that all pieces P are horizontal parallelograms where $\operatorname{height}(P) = 1$ and $\operatorname{width}(P) \leq 1$.

We define an infinite family of *box types* as follows. The box types can be represented by an infinite ternary *box type tree*; see Figure 7. The empty vector [ ] denotes the *basic box type*, which is simply a rectangle of size 2×1. Each box type will be defined as a horizontal parallelogram of height 1. A box type is represented by a vector $[x_1, \ldots, x_d] \in \{-1, 0, +1\}^d$. Given a d-dimensional box type $T := [x_1, \ldots, x_d] \in \{-1, 0, +1\}^d$, we

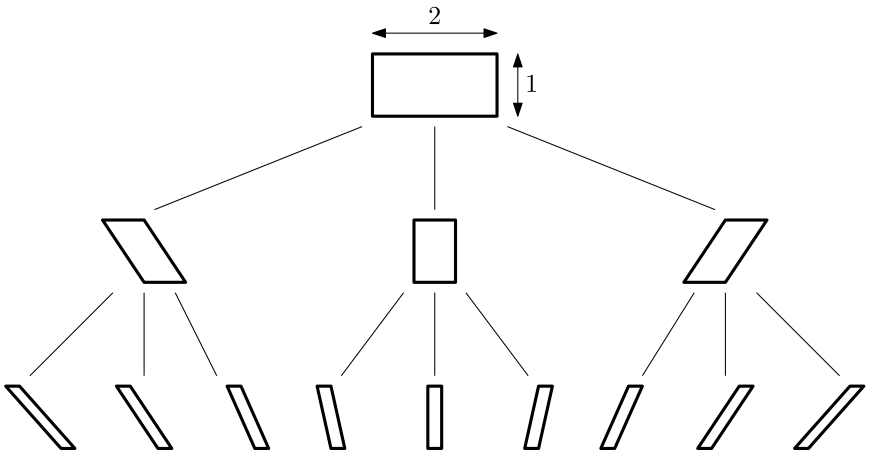


Figure 7: The top three layers of the box type tree.

define $(d+1)$-dimensional box types $T \oplus [x_{d+1}] = [x_1, \ldots, x_d, x_{d+1}]$ for $x_{d+1} \in \{-1, 0, +1\}$, as follows. Let b and t be the bottom and top edges of T. We partition b and t into three equally long segments b_{-1}, b_0, b_{+1} and t_{-1}, t_0, t_{+1}, respectively, in order from left to right. For $x_{d+1} \in \{-1, 0, +1\}$, we then define $T \oplus [x_{d+1}]$ to be the parallelogram spanned by the segments b_0 and $t_{x_{d+1}}$. It follows that a box of type $[x_1, \ldots, x_d]$ has base edges of length $2 \cdot 3^{-d}$ and the other pair of parallel edges have width $2\sum_{i=1}^{d} x_i/3^i$.

Lemma 14. *For every $d \geq 0$ and every line segment s of height 1 and width at most 1, there exists a d-dimensional box type T that can contain s when the lower endpoint of s is placed at the midpoint of the bottom segment of T.*

Proof. We prove the lemma by induction on d. The claim clearly holds for $d = 0$, because the basic box type is a rectangle of size 2×1 and s has height 1 and width at most 1. Suppose that for some dimension $d \geq 0$, there is a box type T that satisfies the lemma. We then partition the bottom and top edges of T, as described in the definition of the box types. We choose $x_{d+1} \in \{-1, 0, +1\}$ such that $t_{x_{d+1}}$ contains the upper endpoint of s. Then $T \oplus [x_{d+1}]$ is a $(d+1)$-dimensional box type that satisfies the claim. $\square$

Lemma 15. *For every horizontal parallelogram P where* $\text{height}(P) = 1$ *and* $\text{width}(P) \leq 1$*, there exists a box type T such that P can be packed into T and* $\text{area}(T) \leq 6\,\text{area}(P)$*.*

Proof. Let ℓ be the length of the horizontal segments of P. Choose $d \geq 0$ as large as possible such that $3^{-d} \geq \ell$; this is possible because $\ell \leq 1$. Let s be a segment of height 1 parallel to the non-horizontal edges of P and consider the d-dimensional box type that contains s as described in Lemma 14. If the top endpoint of s is in the right half of the top edge of T, then we place P to the left of s, i.e., with the right edge of P coincident with s. Otherwise, we place P to the right of s. Since the horizontal segments of T have length $2 \cdot 3^{-d} \geq 2\ell$, it follows that T can contain P. Moreover, by maximality of d, the base edges have length less than 6ℓ, so it follows that $\text{area}(T) \leq 6\,\text{area}(P)$. $\square$

If a piece P can be packed into a box type T and $\text{area}(T) \leq 6\,\text{area}(P)$, as in the lemma, then we say that T *matches* P. We say that a box type T is *suitable* for a piece P if T is an ancestor in the box type tree of a box type that matches P. In particular, P can be packed in a box B of a suitable type, but the area of B may be much more than $6\,\text{area}(P)$. We consider a type T to be an ancestor of itself.

Our algorithm will allocate space of the strip for boxes of the various box types. Each box of type T contains either:

- a piece P that T matches, or
- one, two, or three boxes of types that are children of T in the box type tree.

We now explain the behavior of our algorithm `OnlinePacker` when a new piece P arrives; see Figure 8. Suppose first that there exists a box B_1 that satisfies the following properties.

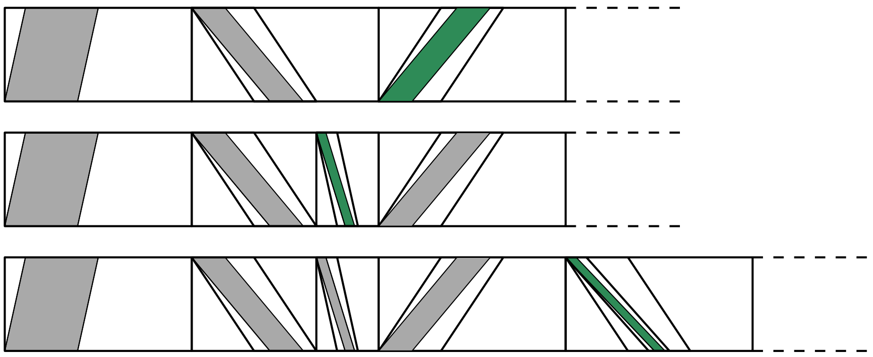


Figure 8: An evolving packing produced by the algorithm. In each case, the green piece has just been placed. In the third step, there was no box suitable for the piece which also had room for it, so we allocated a new basic box.

- The type T_1 of B_1 is suitable for P, i.e., T_1 is an ancestor of a type T_k that matches P in the box type tree. Here, $T_1, \ldots, T_k$ be the path from T_1 to T_k in the box type tree.
- The box B_1 has room for one more box of type T_2.

If more than one such box exist, we choose B_1 as small as possible, i.e., with maximum dimension.

If B_1 exists, we do the following for each $i = 1, \ldots, k-1$: We allocate in B_i an empty box B_{i+1} of type T_{i+1} which is placed in B_i as far left as possible. At last, we place P in the box B_k. Thus, each of the new boxes $B_2, \ldots, B_{k-1}$ will contain a single box, and B_k will contain P.

If such a box B_1 does not exist, we allocate a new box of the basic type and place it as far left in the strip as possible (that is, without overlapping with any already allocated box). We then define it to be our box B_1 and do as explained above; allocating a chain of nested boxes until we get to a type that matches P and place P there.

We say that a d-dimensional box is *near-empty* if there is exactly one $(d+1)$-dimensional box allocated in it. A crucial property of `OnlinePacker` is that it does not produce an excessive number of near-empty boxes, as described in the following lemma.

Figure 9: For any type T it holds that a box of type T can contain two boxes B_1, B_2 of types that are children of T if B_1 and B_2 have the same type or one of them has type $T \oplus [0]$. Here, it is shown for the base type $T := [\,]$.

Lemma 16. *There can be at most two near-empty boxes of each type in a packing produced by `OnlinePacker`.*

Proof. Suppose that there are two near-empty boxes B_1 and B_2 of type T, where B_1 was created first. Let P be the piece that caused B_2 to be allocated. We first see that one of the two boxes, say B_1, must contain a box B_1' of type $T \oplus [-1]$ and the other, B_2, must contain a box B_2' of type $T \oplus [+1]$: If B_1' and B_2' had the same type or one of them had type $T \oplus [+0]$, then they could be packed in the same box of type T; see Figure 9. Hence, P could have been placed in B_1 instead of allocating the new box B_2, contradicting that we allocate new boxes in a smallest possible existing box.

But since B_1' has type $T \oplus [-1]$ and B_2' has type $T \oplus [+1]$, it follows that the next box of type $T \oplus [x]$, for $x \in \{-1, 0, +1\}$, can be placed in B_1 or B_2. Therefore, the algorithm will never allocate a third near-empty box of type T. $\square$

We will now analyze the density under the assumption that there are no near-empty boxes. We shall then reduce the general case to this restricted case.

Lemma 17. *Suppose that `OnlinePacker` has produced a packing of n horizontal parallelograms of height 1 and width at most 1, where there are no near-empty boxes. Then the density of the pieces in the occupied part of the strip is at least $\Omega(n^{1-\log 3})$.*

Proof. We prove that when there are no near-empty boxes, then the density in each basic box is at least $n^{1-\log 3}$, and the lemma follows. So consider a basic box B and let n_B be the number of pieces packed in B. The box B and the boxes allocated in it can be represented as a rooted tree $\mathcal{T}_1$. Here, the root is B and the children of a d-dimensional box are the $(d+1)$-dimensional boxes that it contains, and each leaf is a box that contains a piece. Since there are no near-empty boxes, each internal node in $\mathcal{T}_1$ has two or three children. To get a lower bound on the density in B, we construct a sequence of trees $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_k$, where each tree corresponds to a packing in a basic box. The density of these packings decreases, and we will see in the end that the density in $\mathcal{T}_k$ is at least $\Omega(n^{1-\log 3})$.

The final tree $\mathcal{T}_k$ is balanced in the sense that all leaves are contained in two neighbouring levels d and $d+1$. If a tree $\mathcal{T}_i$ is not balanced, we construct $\mathcal{T}_{i+1}$ in the following way. Let d be the maximum dimension of a box in $\mathcal{T}_i$. We can find two or three leaves of dimension d_u that have the same parent u. If there are three, we remove one of them and the unique piece it contains from the packing, which decreases the density, and then proceed as in the case where there are two, as described in the following. Let u_1, u_2 be the two leaves. Suppose that there is also a leaf v at level $d_v \leq d_u - 2$. Recall that a d-dimensional box has area $2 \cdot 3^{-d}$. Thus, the area of pieces in the boxes u_1, u_2, v combined is at least

$$A := 2 \cdot 2 \cdot 3^{-d_u}/6 + 2 \cdot 3^{-d_v}/6 = 2 \cdot 3^{-(d_u+1)} + 3^{-(d_v+1)}.$$

We now make a replacement argument: We "move" the leafs u_1, u_2, so that they become children of v instead of u, and argue that this only decreases the density. So, we construct a tree T_{i+1} that is similar to $\mathcal{T}_i$, except u is now a leaf and v has two children v_1, v_2. For this to be a conceivable packing, we must remove the pieces that were before in u_1, u_2, v and place new pieces in u, v_1, v_2. We place pieces in u, v_1, v_2 that fills the boxes with density $1/6$. These new pieces have total area

$$A' := 2 \cdot 3^{-d_u+1}/6 + 2 \cdot 2 \cdot 3^{-d_v-1}/6 = 3^{-d_u} + 2 \cdot 3^{-(d_v+2)}.$$

It is now straightforward to verify that since $d_v + 2 \leq d_u$, we have $A' \leq A$. Hence, the density in $\mathcal{T}_{i+1}$ is smaller than the density in $\mathcal{T}_i$.

In the end, we obtain the packing represented by a balanced tree $\mathcal{T}_k$, where all leaves are d- or $(d+1)$-dimensional for some value $d \geq 0$. We then have $d \leq \log n_B + 1$. Therefore, each piece has area at least $2 \cdot 3^{-\log n_B - 1}/6 = \Omega(3^{-\log n})$, and the total area of the pieces is $\Omega(3^{-\log n_B} \cdot n_B) = \Omega(n_B^{1-\log 3}) = \Omega(n^{1-\log 3})$, which is also a lower bound on the density. ◻

We now prove that when the total area of pieces is at least 1, then the density of an arbitrary packing produced by the algorithm, i.e., where there may also be near-empty boxes, is not much smaller.

Lemma 18. *Suppose that* `OnlinePacker` *has packed n horizontal parallelograms of height 1 and width at most 1. If the total area of the pieces is at least 1, the resulting packing has density $\Omega(n^{1-\log 3}/\log n)$.*

Proof. Let A be the area of the strip used by the algorithm, and let Σ be the total area of the pieces. We show that by replacing some boxes and pieces, so that the total area of pieces increases by at most $F = O(\log n)$, we obtain a packing with m pieces, where $m \leq n$, and no near-empty boxes. We then get from Lemma 17 that the resulting packing has density

$$\frac{\Sigma + F}{A} = \Omega(m^{1-\log 3}) = \Omega(n^{1-\log 3}).$$

Using that $\Sigma \geq 1$, it then follows that the original density is

$$\frac{\Sigma}{A} = \frac{\Sigma + F}{A} \cdot \frac{\Sigma}{\Sigma + F} \geq \frac{\Sigma + F}{A} \cdot \frac{\Sigma}{\Sigma + O(\log n)} \geq \frac{\Sigma + F}{A} \cdot \frac{\Sigma}{\Sigma \cdot O(\log n)} = \Omega(n^{1-\log 3}/\log n).$$

Consider a maximal near-empty box B_1, i.e., a near-empty box that is not contained in a larger near-empty box. We remove all boxes contained in B_1 and the pieces they contain and instead place a single piece in B_1 that completely fills B_1. This operation cannot increase the number of pieces, because there was at least one piece contained in B_1 before.

Let d be maximum such that there are at least 2^d near-empty d-dimensional boxes. We first observe that $d \leq \log n$: Each of the near-empty d-dimensional boxes contains a distinct piece. We therefore have $2^d \leq n$, so that $d \leq \log n$.

Let $F_{\leq d}$ be the total area of pieces that we add to eliminate maximal near-empty boxes of dimension at most d and let $F_{>d}$ be the remaining ones, so that $F = F_{\leq d} + F_{>d}$.

In order to bound $F_{\leq d}$, we note that there are 3^i types of i-dimensional boxes, each of which has area $O(3^{-i})$. For each maximal near-empty i-dimensional box, we add a piece of area $O(3^{-i})$. As there are at most two near-empty boxes of each type by Lemma 16, we add boxes of total area $O(3^i \cdot 3^{-i}) = O(1)$ for each $i \leq d$. As $d = O(\log n)$, we have $F_{\leq d} = O(\log n)$.

By similar arguments, we get

$$F_{>d} < \sum_{i=d+1}^{\infty} 2^i \cdot O(3^{-i}) = \sum_{i=d+1}^{\infty} O((2/3)^i) = O(1),$$

so we conclude that $F = O(\log n)$, which finishes the proof. □

5.2.2 Algorithm for horizontal parallelograms of extended width ≤ 1

We now describe an extension of `OnlinePacker` in order to handle horizontal parallelograms of arbitrary height (at most 1) and bounded extended width, to be defined shortly. Here, we partition the pieces into height classes, so that a piece P belongs to class h if $2^{-h-1} < \text{height}(P) \leq 2^{-h}$. For height class h, the base type is a rectangle of size 2×2^{-h}, and in the strip we allocate boxes of this size in which to pack pieces from the height class. We define an infinite ternary box type tree for each height class, exactly as when all the pieces have height 1.

When a piece P arrives, we determine the height class h of P. We extend the non-horizontal segments of P until we obtain a horizontal parallelogram of height exactly 2^{-h}, and we denote the resulting piece P' so that $P \subset P'$. We then define the *extended width* of P as $\text{extwidth}(P) := \text{width}(P')$. Note that since $2^{-h-1} < \text{height}(P)$, we have $\text{height}(P') < 2\,\text{height}(P)$ and $\text{extwidth}(P) = \text{width}(P') < 2\,\text{width}(P)$. We then pack P' (including P) into a minimum suitable box that has already been allocated, if possible, and otherwise allocate a new basic box.

We stack the basic boxes of different height classes if possible. When a new basic box of height class h is created, we stack it on the leftmost pile of basic boxes that has room for it. This choice implies that there can be at most one stack of basic boxes that is less than half full.

Lemma 19. *Suppose that `OnlinePacker` has packed n horizontal parallelograms of width at most $1/2$ and total area more than 2. The resulting packing has density $\Omega(n^{1-\log 3}/\log n)$.*

Proof. Let U be the union of the base boxes (across all height classes), and let A be the area of the part of the strip occupied by the packing. Since the total area of pieces is more than 2, we also have $\text{area}(U) \geq 2$. Suppose first it holds that the density in U is at least $\Omega(n^{1-\log 3}/\log n)$. We have that $\text{area}(A) \leq 2\,\text{area}(U)+2$, since there is at most one stack of base boxes of height less than $1/2$ in the packing. Since $2 < U$, we then also have $\text{area}(A) < 3\,\text{area}(U)$. We can therefore also conclude that the density of the occupied part of the strip is $\Omega(n^{1-\log 3}/\log n)$.

We now prove the density bound in U. Let Σ be the total area of the pieces. For each height class h for which there are some pieces, we feed the algorithm with a rectangle of size 1×2^{-h}. Let F be the total area of these pieces, and we have $F < \sum_{h=0}^{\infty} 2^{-h} = 2$. We therefore have $2\Sigma \geq \Sigma + F$. Let U' be the union of the base boxes in this expanded packing. We then have that the original density is

$$\frac{\Sigma}{\text{area}(U)} \geq \frac{\Sigma + F}{2\,\text{area}(U)} \geq \frac{\Sigma + F}{2\,\text{area}(U')}.$$

Since the area of pieces in each non-empty height class h is at least 2^{-h}, we can now apply Lemma 18 (by scaling the y-coordinates by a factor of 2, we obtain a packing of parallelograms of height 1). Let n_h be the number of pieces in height class h. We get that the density in the base boxes of height 2^{-h} is $\Omega(n_h^{1-\log 3}/\log n_h) = \Omega(n^{1-\log 3}/\log n)$, so this is a bound on the density in all of U'. Hence we have $\frac{\Sigma}{\text{area}\,U} = \Omega(n^{1-\log 3}/\log n)$, and this concludes the proof. □

5.2.3 Algorithm for all horizontal parallelograms

We now describe an extension of `OnlinePacker` that handles horizontal parallelograms of arbitrary height and arbitrary width. We partition the parallelograms into *width classes*. Pieces of width class i are packed in base boxes of width 2^i. Consider a parallelogram P. If $\text{extwidth}(P) \leq 1$, then the width class of P is $i = 1$. Otherwise, P belongs to the width class i such that $2^{i-1} < 2\,\text{extwidth}(P) \leq 2^i$. We handle each width class independently, so that pieces from each class are packed in stacks as described in Section 5.2.2. In particular, each width class i is subdivided into height classes, and we allocate in the strip rectangles of size $2^i \times 1$, where we can place a stack of base boxes of sizes $2^i \times 2^{-h}$, for various values of $h \geq 0$.

When a new piece P arrives, we determine its width class w and height class h. Then, we pack it into a rectangle of size $2^w \times 1$, in a base box of size $2^w \times 2^{-h}$ that has room for it. If no rectangle of that size has room for P, a new rectangle of size $2^w \times 1$ is created and placed as far left in the strip as possible.

Lemma 20. *The competitive ratio of `OnlinePacker` when applied to n horizontal parallelograms of arbitrary width is $O(n^{\log 3-1}\log n)$.*

Proof. After `OnlinePacker` has packed the n parallelograms, we denote the cost of the produced solution as ALG and the optimal offline solution as OPT. We then feed the algorithm with four rectangles of size $2^{i-2} \times 1$ for each width class i for which there are some pieces. We denote the cost of the resulting packing as ALG^+ and the cost of the optimal offline solution as OPT^+. Suppose that the largest width class is class k. We now claim that some piece P has extended width more than 2^{k-2}. If $k > 1$ this holds for any piece in width class k, and if $k = 1$, it follows since we assume that the first piece presented to the algorithm has width 1. We then have $\text{OPT} \geq \text{width}(P) > \text{extwidth}(P)/2 > 2^{k-3}$. Since the added pieces are rectangles of height 1, the optimal packing is similar to the optimal packing for the original instance with the extra pieces added in the end. We therefore have

$$\text{OPT}^+ \leq \text{OPT} + \sum_{i=1}^{k} 2^i < \text{OPT} + 2^{k+1} < 17 \cdot \text{OPT}.$$

Let n_i and Σ_i be the number and total area of the pieces in width class i, respectively. Note that $\Sigma_i > 2^i$, so that we can apply Lemma 19 to each width class (under proper scaling). The bound on the competitive ratio becomes

$$\frac{\text{ALG}}{\text{OPT}} \leq \frac{\text{ALG}^+}{\text{OPT}^+/17} \leq \frac{17\sum_{i=1}^{k} n_i^{\log 3-1}\log n_i \cdot \Sigma_i}{\text{OPT}^+} \leq \frac{17 n^{\log 3-1}\log n \cdot \Sigma}{\Sigma} = O(n^{\log 3-1}\log n).$$

$\square$

5.2.4 Algorithm for all convex pieces

We now describe the extension of `OnlinePacker` to handle arbitrary convex pieces; see Figure 10. When a new piece P arrives, we find a horizontal parallelogram P' such that $P \subset P'$, $\text{area}(P') \leq 2\,\text{area}(P)$ and $\text{width}(P') \leq 2\,\text{width}(P)$; then we apply `OnlinePacker` to the parallelogram P' (with P inside).

We define P' as follows. Let ℓ_b and ℓ_t be the lower and upper horizontal tangent to P, respectively, and let $c_b \in P \cap \ell_b$ and $c_t \in P \cap \ell_t$. Let ℓ_l and ℓ_r be the left and right tangent to P parallel to the segment $c_b c_t$, respectively. We then define P' to be the horizontal parallelogram enclosed by the lines $\ell_b, \ell_t \ell_l, \ell_r$.

It is straightforward to check that $\text{area}(P') \leq 2\,\text{area}(P)$, because P is convex. Now we prove that $\text{width}(P') \leq 2\,\text{width}(P)$. Define L to be the length of the horizontal edges of P' and note that $\text{width}(P') = \text{width}(c_b c_t) + L$. On the other hand, $\text{width}(P) \geq \max\{\text{width}(c_b c_t), L\}$, and this proves the claim.

In the proof of Lemma 20 we only bound OPT using the width of the widest piece and the total area Σ. With respect to both width and area, the value for an arbitrary convex pieces P is at least $1/2$ of that of the containing parallelogram P'. Therefore, the bound on the competitive ratio carries over up to constant factors when the algorithm is applied to the parallelograms P', as stated in the following.

Theorem 5. *There exists an algorithm for* Strip-Packing *with competitive ratio $O(n^{\log 3-1}\log n)$, where n is the number of pieces.*

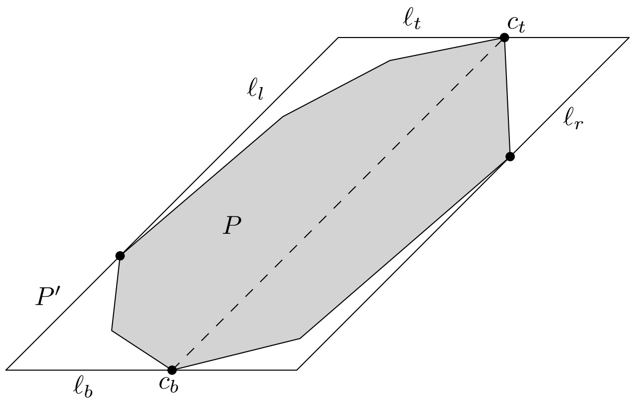


Figure 10: The horizontal parallelogram P' has area at most twice that of the piece P.

6 Constant-factor approximations for offline packing

In this section we show how to obtain offline approximation algorithms for the packing problems of Theorem 2.

Theorem 6. *There are polynomial-time offline algorithms for the following packing problems:*

(a) Offline-Strip-Packing*, 32.7-approximation algorithm.*

(b) Offline-Bin-Packing*, 10.1-approximation algorithm if the diameter of all pieces is bounded by 1/10.*

(c) Offline-Perimeter-Packing*, 8.9-approximation algorithm.*

(d) Offline-Square-Packing[1/10]*, in particular, every set of convex polygons of diameter and total area at most 1/10 can be packed into the unit square.*

Proof. Alt, de Berg, and Knauer [5, 6] presented an algorithm that packs any set $\mathcal{P}$ of convex polygons (with n vertices in total, using $O(n \log n)$ time) into an axis-aligned rectangular container B such that $\text{area}(B) \leq 23.78 \cdot \text{OPT}$, where OPT is the minimum area of any axis-aligned rectangular container for $\mathcal{P}$. As an intermediate step, they obtain a collection of rectangular mini-containers that together contain all objects of $\mathcal{P}$. Let $w_{\max}$ and $h_{\max}$ denote the maximum width and height among all objects of $\mathcal{P}$, respectively. For some fixed constants $c > 0, \alpha \in (0,1)$, each mini-container has width $(c+1)w_{\max}$ and a height h_i where $h_i := \alpha^i h_{\max}$ for some appropriate i. The total area A_C of all mini-containers can be bounded by

$$A_C \leq (1 + 1/c) \cdot \left[\frac{2}{\alpha} \cdot \text{area}(\mathcal{P}) + \frac{c + 2/\alpha}{1-\alpha} \cdot h_{\max} w_{\max}\right]. \tag{1}$$

In order to prove (a), (b), (c), and (d), we repeatedly make use of the mini-containers and this equation.

We first consider strip packing and prove (a). When packing a set of convex polygons $\mathcal{P}$ into a horizontal strip of height 1, the minimal width OPT-W is at least $\max\{w_{\max}, \text{area}(\mathcal{P})\}$. Therefore,

$$A_C \leq (1 + 1/c) \cdot \left[\frac{2}{\alpha} + \frac{c + 2/\alpha}{1-\alpha}\right] \cdot \text{OPT-W}.$$

We group the mini-containers greedily into stacks of height at most 1, which we place in the strip. Note that all but possibly one stack have a height of at least 1/2; otherwise two of these should have been put together. Therefore, the number of stacks is at most $\frac{2A_C}{(c+1)w_{\max}} + 1$. Together with $\alpha := 0.545$ and $c := 2.2$, this translates into a width of

$$2A_C + (c+1)w_{\max} \leq \left(2 \cdot (1 + 1/c) \cdot \left[\frac{2}{\alpha} + \frac{c + 2/\alpha}{1-\alpha}\right] + c + 1\right) \cdot \text{OPT-W} < 32.7 \cdot \text{OPT-W}.$$

We now turn our attention to (d), and our findings will later be used to prove (b). Let S denote the unit square in which we want to pack a given set $\mathcal{P}$ of convex polygons, each of diameter at most δ. We choose $\alpha := 1/2$ and choose the width of the mini-containers to be 1, i.e., $(c+1)w_{\max} = 1$.[2] We stack all mini-containers on top of each other. Suppose that the total height of the mini-containers exceeds 1. We prove that the total area of pieces is then more than the constant $\rho := (1-5\delta)(1-2\delta)/4$. For $\delta := 1/10$, we get $\rho := 1/10$, and the claim in the theorem follows.

We call a mini-container *full* if the bounding box of all contained pieces has width more than $1-\delta$; otherwise there is room for a further piece. A mini-container that is not full is called *near-empty.* We can pack the pieces such that in each height class, there is at most one near-empty mini-container. Therefore, the total height of near-empty mini-containers is at most $\sum_i h_i = h_{\max} \sum_i \alpha^i \leq \delta/(1-\alpha) = 2\delta$. Since the mini-containers (full and near-empty) have a total height of more than 1, the full mini-containers in S have a total height of more than $1-2\delta$, and a total area of more than $1-2\delta$.

Let B be the bounding box of a full mini-container of height h_i (and area h_i) and denote the set of contained polygons by $\mathcal{P}'$. By [5, 6], we obtain

$$(1-\delta)h_i \leq \operatorname{area}(B) \leq 2/\alpha \cdot \big(\operatorname{area}(\mathcal{P}') + h_i w_{\max}\big) \leq 4\big(\operatorname{area}(\mathcal{P}') + h_i\delta\big)\,.$$

Consequently, $\operatorname{area}(\mathcal{P}') \geq (1-5\delta)h_i/4$ and thus the density in any full mini-container is at least $(1-5\delta)/4$. As the total area of the full mini-containers in S is more than $1-2\delta$, the total area packed into S is more than $\rho := (1-5\delta)(1-2\delta)/4$.

We use the described algorithm for square packing to prove statement (b). By the above strategy, if $\operatorname{area}(\mathcal{P}) \leq 1/\rho$, where $\rho := (1-5\delta)(1-2\delta)/4$, then the algorithm will use only one bin, so the packing achieves the optimum in this case. We may therefore assume $\operatorname{area}(\mathcal{P}) > 1/\rho$. We can guarantee a density of at least ρ in all but one bin. Consequently, the number of bins is at most $\operatorname{area}(\mathcal{P})/\rho + 1$. Clearly, the number of bins in the optimal solution is at least $\operatorname{area}(\mathcal{P})$. Therefore, the approximation ratio is at most

$$\frac{\operatorname{area}(\mathcal{P})/\rho + 1}{\operatorname{area}(\mathcal{P})} \leq 1/\rho + \rho.$$

Using $\delta := 1/10$ yields the ratio 10.1 as stated in the theorem.

We finally prove (c). In order to obtain a bounding box with small perimeter, we consider the mini-containers in a greedy fashion (from largest to smallest height) and pack them on top of each other into stacks of height at most $H := \sqrt{A_C} + h_{\max}$ and width $(c+1)w_{\max}$. Clearly the height of each stack except possibly the last one is at least $\sqrt{A_C}$. Consequently, the number of stacks is at most $\frac{\sqrt{A_C}}{(c+1)w_{\max}} + 1$. Hence, we obtain a bounding box with perimeter of at most $4\sqrt{A_C} + 2h_{\max} + 2(c+1)w_{\max}$. Because the optimal perimeter OPT-P is at least $\max\{2w_{\max} + 2h_{\max}, 4\sqrt{\operatorname{area}(\mathcal{P})}\}$, we have $h_{\max}w_{\max} \leq \text{OPT-P}^2/8$ and $\operatorname{area}(\mathcal{P}) \leq \text{OPT-P}^2/16$. We then get from Equation (1) that

$$A_C \leq (1 + {}^1\!/_c) \cdot \left[\frac{2}{\alpha} \cdot \frac{1}{16} + \frac{c + {}^2\!/_\alpha}{1-\alpha} \cdot \frac{1}{8}\right] \text{OPT-P}^2.$$

Finally, choosing $\alpha := 0.5$ and $c := 1.06$, we obtain

$$4\sqrt{A_C} + 2h_{\max} + 2(c+1)w_{\max} \leq \left(4\sqrt{(1 + {}^1\!/_c) \cdot \left[\frac{2}{\alpha} \cdot \frac{1}{16} + \frac{c + {}^2\!/_\alpha}{1-\alpha} \cdot \frac{1}{8}\right]} + 1 + c\right) \cdot \text{OPT-P} < 8.9 \cdot \text{OPT-P}.$$

This completes the proof. □

[2] In fact, we can choose α depending on δ to get a denser packing. It turns out that $\alpha := 1 - \frac{\sqrt{3\delta^3 - 4\delta^2 + \delta}}{1-\delta}$ is the optimal value, but we stick to $\alpha := 1/2$ to keep the analysis simpler.

Acknowledgements

We thank Shyam Narayanan for improving the upper bound algorithm in Theorem 3 to the asymptotically tight upper bound of $O(\sqrt{n})$.

We highly appreciate the BARC espresso machine, the comfy couches in the Creative Room, and the cucumbers. We also thank Linda's travel grant, Denmark for opening the borders in the right moment after a COVID-19 lockdown, and the lenient train officer who trusted the great authority of Mikkel's letter of invitation, allowing for travelling to Copenhagen and making this work possible.

References

[1] Mikkel Abrahamsen and Lorenzo Beretta. "Online Packing to Minimize Area or Perimeter". In: *37th International Symposium on Computational Geometry (SoCG 2021)*. 2021, 6:1–6:15. DOI: 10.4230/LIPIcs.SoCG.2021.6.

[2] Hee-Kap Ahn and Otfried Cheong. "Aligning two convex figures to minimize area or perimeter". In: *Algorithmica* 62.1-2 (2012), pp. 464–479. DOI: 10.1007/s00453-010-9466-1.

[3] Susanne Albers. "Online scheduling". In: *Introduction to scheduling*. CRC Press, 2009, pp. 71–98.

[4] Helmut Alt. "Computational Aspects of Packing Problems". In: *Bulletin of the EATCS* 118 (2016). URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/390.

[5] Helmut Alt, Mark de Berg, and Christian Knauer. "Approximating minimum-area rectangular and convex containers for packing convex polygons". In: *JoCG* 8.1 (2017), pp. 1–10. DOI: 10.20382/jocg.v8i1a1.

[6] Helmut Alt, Mark de Berg, and Christian Knauer. "Corrigendum to: Approximating minimum-area rectangular and convex containers for packing convex polygons". In: *JoCG* 11.1 (2020), pp. 653–655. DOI: 10.20382/jocg.v11i1a26.

[7] Helmut Alt, Otfried Cheong, Ji-won Park, and Nadja Scharf. "Packing 2D Disks into a 3D Container". In: *International Workshop on Algorithms and Computation (WALCOM 2019)*. 2019, pp. 369–380. DOI: 10.1007/978-3-030-10564-8_29.

[8] Helmut Alt and Ferran Hurtado. "Packing Convex Polygons into Rectangular Boxes". In: *18th Japanese Conference on Discrete and Computational Geometry (JCDCGG 2000)*. 2000, pp. 67–80.

[9] Brenda S. Baker and Jerald S. Schwarz. "Shelf algorithms for two-dimensional packing problems". In: *SIAM Journal on Computing* 12.3 (1983), pp. 508–525. DOI: 10.1137/0212033.

[10] Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam SchlÖter, Kevin Schewior, and Leen Stougie. "Tight Bounds for Online TSP on the Line". In: *ACM Transactions on Algorithms* 17.1 (2021). DOI: 10.1145/3422362.

[11] Brian Brubach. "Improved Bound for Online Square-into-Square Packing". In: *Approximation and Online Algorithms - 12th International Workshop (WAOA 2014)*. 2014, pp. 47–58. DOI: 10.1007/978-3-319-18263-6_5.

[12] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. "A Review of Machine Scheduling: Complexity, Algorithms and Approximability". In: *Handbook of Combinatorial Optimization: Volume1–3*. Ed. by Ding-Zhu Du and Panos M. Pardalos. 1998, pp. 1493–1641. DOI: 10.1007/978-1-4613-0303-9_25.

[13] Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. "Approximation and online algorithms for multidimensional bin packing: A survey". In: *Computer Science Review* 24 (2017), pp. 63–79. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2016.12.001.

[14] Fan Chung and Ron Graham. "Efficient packings of unit squares in a large square". In: *Discrete & Computational Geometry* (2019).

[15] János Csirik and Gerhard J. Woeginger. "On-line packing and covering problems". In: *Online Algorithms: The State of the Art*. Ed. by Amos Fiat and Gerhard J. Woeginger. Springer, 1998, pp. 147–177. ISBN: 978-3-540-68311-7. DOI: 10.1007/BFb0029568.

[16] János Csirik and Gerhard J. Woeginger. "Shelf Algorithms for On-Line Strip Packing". In: *Information Processing Letters* 63.4 (1997), pp. 171–175. DOI: 10.1016/S0020-0190(97)00120-8.

[17] Leah Epstein and Rob van Stee. "Multidimensional packing problems". In: *Handbook of Approximation Algorithms and Metaheuristics*. Ed. by Teofilo F. Gonzalez. Second. Chapman and Hall/CRC, 2018, pp. 553–570. DOI: 10.1201/9781351236423-31.

[18] Paul Erdős and Ron Graham. "On packing squares with equal squares". In: *Journal of Combinatorial Theory, Series A* 19.1 (1975), pp. 119–123. DOI: 10.1016/0097-3165(75)90099-0.

[19] Sándor P. Fekete and Hella-Franziska Hoffmann. "Online Square-into-Square Packing". In: *Algorithmica* 77.3 (2017), pp. 867–901. DOI: 10.1007/s00453-016-0114-2.

[20] Amos Fiat and Gerhard J. Woeginger. "Competitive analysis of algorithms". In: *Online Algorithms: The State of the Art*. Ed. by Amos Fiat and Gerhard J. Woeginger. 1998, pp. 1–12. DOI: 10.1007/BFb0029562.

[21] Amos Fiat and Gerhard J. Woeginger. "On-line scheduling on a single machine: Minimizing the total completion time". In: *Acta Informatica* 36.4 (1999), pp. 287–293. DOI: 10.1007/s002360050162.

[22] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey". In: *Discrete Optimization II*. Ed. by P.L. Hammer, E.L. Johnson, and B.H. Korte. Vol. 5. Annals of Discrete Mathematics. 1979, pp. 287–326. DOI: 10.1016/S0167-5060(08)70356-X.

[23] Xin Han, Francis Y. L. Chin, Hing-Fung Ting, Guochuan Zhang, and Yong Zhang. "A new upper bound 2.5545 on 2D Online Bin Packing". In: *ACM Transactions on Algorithms* 7.4 (2011), 50:1–50:18. DOI: 10.1145/2000807.2000818.

[24] Xin Han, Kazuo Iwama, Deshi Ye, and Guochuan Zhang. "Strip Packing vs. Bin Packing". In: *Algorithmic Aspects in Information and Management (AAIM 2007)*. 2007, pp. 358–367. DOI: 10.1007/978-3-540-72870-2_34.

[25] Xin Han, Kazuo Iwama, and Guochuan Zhang. "Online removable square packing". In: *Theory of Computing Systems* 43.1 (2008), pp. 38–55. DOI: 10.1007/s00224-007-9039-0.

[26] Janusz Januszewski and Marek Lassak. "On-line packing sequences of cubes in the unit cube". In: *Geometriae Dedicata* 67.3 (1997), pp. 285–293. DOI: 10.1023/A:1004953109743.

[27] A. Kosiński. "A proof of an Auerbach-Banach-Mazur-Ulam theorem on convex bodies". In: 4 (1957), pp. 216–218. DOI: 10.4064/cm-4-2-216-218.

[28] Marek Lassak and Jixian Zhang. "An on-line potato-sack theorem". In: *Discrete & Computational Geometry* 6.1 (1991), pp. 1–7. DOI: 10.1007/BF02574670.

[29] Hyun-Chan Lee and Tony C. Woo. "Determining in Linear Time the Minimum Area Convex Hull of Two Polygons". In: *IIE Transactions* 20.4 (1988), pp. 338–345. DOI: 10.1080/07408178808966189.

[30] Boris D. Lubachevsky and Ronald L. Graham. "Dense packings of congruent circles in rectangles with a variable aspect ratio". In: *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*. Ed. by Boris Aronov, Saugata Basu, János Pach, and Micha Sharir. 2003, pp. 633–650. DOI: 10.1007/978-3-642-55566-4_28.

[31] Boris D. Lubachevsky and Ronald L. Graham. "Minimum perimeter rectangles that enclose congruent non-overlapping circles". In: *Discrete Mathematics* 309.8 (2009), pp. 1947–1962. ISSN: 0012-365X. DOI: 10.1016/j.disc.2008.03.017.

[32] R. Daniel Mauldin. *The Scottish Book: Mathematics from the Scottish Café, with Selected Problems from the New Scottish Book*. 2015. DOI: 10.1007/978-3-319-22897-6.

[33] Victor J. Milenkovic. "Rotational polygon containment and minimum enclosure using only robust 2D constructions". In: *Computational Geometry* 13.1 (1999), pp. 3–19. ISSN: 0925-7721. DOI: 10.1016/S0925-7721(99)00006-1.

[34] Victor J. Milenkovic. "Translational polygon containment and minimal enclosure using linear programming based restriction". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing (STOC 1996)*. 1996, pp. 109–118. DOI: 10.1145/237814.237840.

[35] Victor J. Milenkovic and Karen Daniels. “Translational polygon containment and minimal enclosure using mathematical programming”. In: *International Transactions in Operational Research* 6.5 (1999), pp. 525–554. DOI: 10.1111/j.1475-3995.1999.tb00171.x.

[36] J.W. Moon and L. Moser. “Some packing and covering theorems”. In: *Colloquium Mathematicum*. Vol. 17. 1. 1967, pp. 103–110. DOI: 10.4064/cm-17-1-103-110.

[37] Dongwoo Park, Sang Won Bae, Helmut Alt, and Hee-Kap Ahn. “Bundling three convex polygons to minimize area or perimeter”. In: *Computational Geometry* 51 (2016), pp. 1–14. ISSN: 0925-7721. DOI: 10.1016/j.comgeo.2015.10.003.

[38] Kirk Pruhs, Jiří Sgall, and Eric Torng. “Online scheduling”. In: *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Ed. by Joseph Y-T. Leung. 2004.

[39] Jiří Sgall. “On-line scheduling”. In: *Online Algorithms: The State of the Art*. Ed. by Amos Fiat and Gerhard J. Woeginger. 1998, pp. 196–231. DOI: 10.1007/BFb0029570.

[40] E. Specht. “High density packings of equal circles in rectangles with variable aspect ratio”. In: *Computers & Operations Research* 40.1 (2013), pp. 58 –69. ISSN: 0305-0548. DOI: 10.1016/j.cor.2012.05.011.

[41] Rob van Stee. “SIGACT news online algorithms column 20: the power of harmony”. In: *SIGACT News* 43.2 (2012), pp. 127–136. DOI: 10.1145/2261417.2261440.

[42] Rob van Stee. “SIGACT News Online Algorithms Column 26: Bin packing in multiple dimensions”. In: *SIGACT News* 46.2 (2015), pp. 105–112. DOI: 10.1145/2789149.2789167.

[43] Deshi Ye, Xin Han, and Guochuan Zhang. “A note on online strip packing”. In: *Journal of Combinatorial Optimization* 17.4 (2009), pp. 417–423. DOI: 10.1007/s10878-007-9125-x.

Appendix E

FOCS: Locally Uniform Hashing

Locally Uniform Hashing

Ioana O. Bercea[1], Lorenzo Beretta[2], Jonas Klausen[2], Jakob Bæk Tejs Houen[2], and Mikkel Thorup[2]

[1]IT University of Copenhagen , {iobe}@itu.dk
[2]University of Copenhagen, {beretta, jokl, jakn, mthorup}@di.ku.dk

Abstract

Hashing is a common technique used in data processing, with a strong impact on the time and resources spent on computation. Hashing also affects the applicability of theoretical results that often assume access to (unrealistic) uniform/fully-random hash functions. In this paper, we are concerned with designing hash functions that are practical and come with strong theoretical guarantees on their performance.

To this end, we present tornado tabulation hashing, which is simple, fast, and exhibits a certain full, local randomness property that provably makes diverse algorithms perform almost as if (abstract) fully-random hashing was used. For example, this includes classic linear probing, the widely used HyperLogLog algorithm of Flajolet, Fusy, Gandouet, Meunier [AOFA'97] for counting distinct elements, and the one-permutation hashing of Li, Owen, and Zhang [NIPS'12] for large-scale machine learning. We also provide a very efficient solution for the classical problem of obtaining fully-random hashing on a fixed (but unknown to the hash function) set of n keys using $O(n)$ space. As a consequence, we get more efficient implementations of the splitting trick of Dietzfelbinger and Rink [ICALP'09] and the succinct space uniform hashing of Pagh and Pagh [SICOMP'08].

Tornado tabulation hashing is based on a simple method to systematically break dependencies in tabulation-based hashing techniques.

1 Introduction

The generic goal of this paper is to create a practical hash function that provably makes important algorithms behave almost as if (unrealistic) fully random hashing was used. By practical, we mean both simple to implement and fast. Speed is important because hashing is a common inner loop for data processing. Suppose for example that we want to sketch a high-volume data stream such as the packets passing a high-end Internet router. If we are too slow, then we cannot handle the stream at all. Speed matters, also within constant factors.

The use of weak hash functions is dangerous, not only in theory but also in practice. A good example is the use of classic linear hashing. Linear hashing is 2-independent and Mitzenmacher and Vadhan [27] have proved that, for some applications, 2-independent hashing performs as well as fully random hashing if the input has enough entropy, and indeed this often works in practice. However, a dense set has only 1 bit of entropy per element, and [36, 30] have shown that with a linear hashing scheme, if the input is a dense set (or more generally, a dense subset of an arithmetic sequence), then linear probing becomes extremely unreliable and the expected probe length[1] increases from constant to $\Omega(\log n)$. This is also a practical problem because dense subsets may occur for many reasons. However, if the system is only tested on random inputs, then we may not discover the problem before deployment.

The issue becomes even bigger with, say, sampling and estimation where we typically just trust our estimates with no knowledge of the true value. We may never find out that our estimates are bad. With 2-independent hashing, we get the right variance, but not exponential concentration. Large errors can happen way too often, yet not often enough to show up in a few tests. This phenomenon is demonstrated on synthetic data in [2] and on real-world data in [1]. All this shows the danger of relying on weak hash functions without theoretical guarantees for all possible inputs, particularly for online systems where we cannot just change the hash function if the input is bad, or in situations with estimates whose quality cannot be verified. One could instead, perhaps, use hash functions based on cryptographic assumptions, but the hash function that we propose here is simple to implement, fast, and comes with strong unconditional guarantees.

In this paper, we introduce *tornado tabulation hashing.* A tornado tabulation hash function $h : \Sigma^c \to \mathcal{R}$ requires $O(c|\Sigma|)$ space and can be evaluated in $O(c)$ time using, say, $2c$ lookups in tables with $|\Sigma|$ entries plus some simple AC^0 operations (shifts, bit-wise xor, and assignments). As with other tabulation schemes, this is very fast when Σ is small enough to fit in fast cache, e.g., for 32-bit keys divided into $c = 4$ characters of 8 bits (namely, $|\Sigma| = 2^8$), the speed is similar to that of evaluating a degree-2 polynomial over a Mersenne prime field.

Tornado hashing has the strong property that if we hash a set of $|\Sigma|/2$ keys, then with high probability, the hash values are completely independent. For when we want to handle many more keys, e.g., say $|\Sigma|^3$ (as is often the case when Σ is small), tornado tabulation hashing offers a certain *local uniformity* that provably makes a diverse set of algorithms behave almost as if the hashing was fully random on all the keys. The definition of local uniformity is due to Dahlgaard, Knudsen, Rotenberg, and Thorup [10]. The definition is a bit complicated, but they demonstrate how it applies to the widely used HyperLogLog algorithm of the Flajolet, Fusy, Gandouet, Meunier [18] for counting distinct elements in data streams, and the One-Permutation Hashing of Li, Owen, and Zhang [26] used for fast set similarity. They conclude that the estimates obtained are only a factor

[1]The probe length is defined as the number of contiguous cells probed to answer a query of a linear probing hash table.

$1 + o(1)$ worse than if fully-random hashing was used. Interestingly, [10] proves this in a high-level black-box manner. Loosely speaking, the point is that the algorithm using locally uniform hashing behaves as well as the same algorithm using fully-random hashing on a slightly worse input.

As a new example, we will demonstrate this on linear probing. Knuth's original 1963 analysis of linear probing [25], which started the field of algorithms analysis, showed that with fully-random hashing and load $(1 - \varepsilon)$, the expected probe length is $(1 + 1/\varepsilon^2)/2$. From this, we conclude that tornado tabulation hashing yields expected probe length $(1 + o(1))(1 + 1/\varepsilon^2)/2$, and we get that without having to reconsider Knuth's analysis.

For contrast, consider the work on linear probing with k-independent hashing. Pagh, Pagh, Ružić [29] showed that 5-independence is enough to obtain a bound of $O(1/\varepsilon^{13/6})$ on the expected probe length. This was further improved to $O(1/\varepsilon^2)$ by Pătraşcu and Thorup [31], who achieved the optimal $O(1/\varepsilon^2)$. They matched Knuth's bound modulo some large constants hidden in the O-notation and needed a very different analysis.

In practice, the guarantee that we perform almost like fully-random hashing means that no set of input keys will lead to substantially different performance statistics. Thus, if we tested an online system on an arbitrary set of input keys, then we do not have to worry that future input keys will degrade the performance statistics, not even by a constant factor.

The definition of local uniformity is due to Dahlgaard, Knudsen, Rotenberg, and Thorup [10]. They did not name it as an independent property, but they described it as a property of their new hashing scheme: mixed tabulation hashing. However, the local uniformity of mixed tabulation assumes table size $|\Sigma| \to \infty$, but the speed of tabulation hashing relies on $|\Sigma|$ being small enough to fit in fast cache and all reported experiments use 8-bit characters (see [31, 2, 1, 11]). However, none of the bounds from [10] apply to 8 or even 16-bit characters, e.g., they assume $O(\log |\Sigma|)^c < |\Sigma|$. Our new scheme avoids the exponential dependence on c, and we get explicit error probability bounds that are meaningful, not just in theory, but also for practice with tables in fast cache.

For when we want full randomness on more keys than fit in fast cache, we could, as above, increase $|\Sigma|$ in all $O(c)$ lookup tables. In this paper, we show that it suffices to augment the in-cache tornado hashing with just 2 lookups in tables of size $2n$ to get full randomness on n keys with high probability. This would work perfectly inside a linear space algorithm assuming fully-random hashing, but it also leads to more efficient implementations of the spitting trick of Dietzfelbinger and Rink [13] and the succinct space uniform hashing of Pagh and Pagh [28].

In Section 1.1 we define our hash function, and in Section 1.2 we present our technical results, including the definition of local uniformity. In Section 1.3, we discuss more explicitly how our work compares to mixed tabulation and explain some of our techniques in comparison. In Section 1.4 we discuss several applications. In Section 1.5, we relate our work to previous work on achieving highly independent hash functions. Finally, in Section 1.6 we discuss how tornado tabulation can be employed to improve the so-called "splitting trick" and succinct uniform hashing.

1.1 Tornado tabulation hashing

Simple tabulation hashing. We first introduce our main building block, which is the simple tabulation hash function dating back to at least Zobrist [38] and Wegman and Carter [37]. Throughout the paper, we will consider keys to come from the universe Σ^c and hash values to be in $\mathcal{R} = [2^r]$. More concretely, we interpret a key x as the concatenation of c *characters* $x_1 \dots x_c$

from Σ. We then say that a function $h : \Sigma^c \longrightarrow \mathcal{R}$ is a *simple tabulation* hash function if

$$h(x) = T_1[x_1] \oplus \cdots \oplus T_c[x_c]$$

where, for each $i = 1 \ldots c$, $T_i : \Sigma \longrightarrow \mathcal{R}$ is a fully-random function stored as a table.

We think of c as a small constant, e.g., $c = 4$, for 32-bit keys divided into 8-bit characters, yet we will make the dependence on c explicit. We assume that both keys and hash values fit in a single word and that $|\Sigma| \geqslant 2^8$.

Tornado tabulation hashing. To define a tornado tabulation hash function h, we use several simple tabulation hash functions. A tornado tabulation function has a number d of *derived* characters. Think of d as, say, c or $2c$. It will later determine our error probability bounds. We will always assume $d = O(c)$ so that d characters from Σ can be represented in $O(1)$ words of memory (since a key from Σ^c fits in a single word).

For each $i = 0, \ldots, d$, we let $\widetilde{h}_i : \Sigma^{c+i-1} \longrightarrow \Sigma$ be a simple tabulation hash function. Given a key $x \in \Sigma^c$, we define its *derived key* $\widetilde{h}(x) \in \Sigma^{c+d}$ as $\widetilde{x} = \widetilde{x}_1 \cdots \widetilde{x}_{c+d}$, where

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i < c \\ x_c \oplus \widetilde{h}_0(\widetilde{x}_1 \cdots \widetilde{x}_{c-1}) & \text{if } i = c \\ \widetilde{h}_{i-c}\,(\widetilde{x}_1 \cdots \widetilde{x}_{i-1}) & \text{if } i > c. \end{cases} \tag{1}$$

We note that each of the d derived characters $\widetilde{x}_{c+1}, \ldots, \widetilde{x}_{c+d}$ is progressively defined by applying a simple tabulation hash function to all its preceding characters in the derived key $\widetilde{x}$. Hence, the name tornado tabulation. The step by which we obtain $\widetilde{x}_c$ corresponds to the twist from [32]. By Observation 1.1. in [32], $x_1 \ldots x_c \mapsto \widetilde{x}_1 \ldots \widetilde{x}_c$ is a permutation, so distinct keys have distinct derived keys. Finally, we have a simple tabulation hash function $\widehat{h} : \Sigma^{c+d} \longrightarrow \mathcal{R}$, that we apply to the derived key. The *tornado tabulation* hash function $h : \Sigma^c \longrightarrow \mathcal{R}$ is then defined as $h(x) = \widehat{h}(\widetilde{x})$.

Implementation. The simplicity of tornado tabulation is apparent from its C implementation below. In the code, we fold tornado's lookup tables together so we can implement them using $c + d$ character tables $\Sigma \to \Sigma^{d+1} \times \mathcal{R}$. Elements of $\Sigma^{d+1} \times \mathcal{R}$ are just represented as w-bits numbers. For memory alignment and ease of implementation, we want w to be a power of two such as 64 or 128.

We now present a C-code implementation of tornado tabulation for 32-bit keys, with $\Sigma = [2^8]$, $c = 4$, $d = 4$, and $\mathcal{R} = [2^{24}]$. Besides the key `x`, the function takes as input an array of $c + d$ tables of size $|\Sigma|$, all filled with independently drawn 64-bit values.

```
INT32 Tornado(INT32 x, INT64[8][256] H) {
        INT32 i; INT64 h=0; INT8 c;
        for (i=0;i<3;i++) {
                c=x;
                x>>=8;
                h^=H[i][c];}
        h^=x;
        for (i=3;i<8;i++) {
                c=h;
                h>>=8;
                h^=H[i][c];}
        return ((INT32) h);}
```

Speed. As we can see in the above implementation, tornado hashing uses $c + d$ lookups and $O(c + d)$ simple AC^0 operations. The speed of tabulation hashing depends on the tables fitting in fast cache which means that Σ should not be too big. In the above code, we used $\Sigma = [2^8]$, as in all previously reported experiments with tabulation hashing. (see [31, 2, 1, 11]).

The speed of tabulation schemes is incomparable to that of polynomial methods using small space but multiplication. Indeed, the ratio between the cost of cache lookups and multiplication depends on the architecture. In line with previous experiments, we found our tornado implementation for 32-bit keys to be as fast as a degree-2 polynomial over a Mersenne prime ($2^{89} - 1$) field.

We note that our implementation for the random table `H` only needs a pointer to an area filled with "fixed" random bits, and it could conceivably be made much faster if we instead of cache had access to random bits stored in simple read-only memory (ROM or EPROM).

1.2 Theoretical Results

The main aim of our paper is to prove that, with high probability (whp), a tornado tabulation hash function is fully random on some set X of keys. The challenge is to characterize for which kinds of sets we can show such bounds.

Full randomness for fixed keys. We begin with a simpler result that follows directly from our main technical theorem. In this case, the set X of keys is fixed.

Theorem 1. *Let $h : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters. For any fixed $X \subseteq \Sigma^c$, if $|X| \leqslant |\Sigma|/2$, then h is fully random on X with probability at least*

$$1 - 7|X|^3(3/|\Sigma|)^{d+1} - 1/2^{|\Sigma|/2} .$$

With $c, d = O(1)$, Theorem 1 gives an $O(|\Sigma|)$ space hash function that can be evaluated in constant time and, with high probability, will be fully random for any fixed set X of at most $|\Sigma|/2$ keys. This is asymptotically tight as we need $|X|$ random hash values to get this randomness.

The random process behind the error probability that we get will be made clear in the next paragraph. We note here that, since $|X| \leqslant |\Sigma|/2$, we have that $7|X|^3(3/|\Sigma|)^{d+1} \leqslant 24(3/|\Sigma|)^{d-2}$. With $|\Sigma| \geqslant 2^8$, the bound is below $1/300$ for $d = 4$, and decreases rapidly for larger d. For $c \geqslant 4$, if we set $d = 2c$, we get an error probability below $1/u$ where $u = |\Sigma|^c$ is the size of the universe. We can get error probability $1/u^\gamma$ for any constant γ with $d = O(c)$, justifying this assumption on d.

Linear independence. The general structure of our results is to identify some error event such that (1) if the event does not occur, then the hash function will be fully random on X, and (2) the error event itself happens with low probability. The error event that we consider in Theorem 1 is inherent to all tabulation-based hashing schemes. Namely, consider some set Y of keys from some universe Σ^b. We say that Y is *linearly independent* if and only if, for every subset $Y' \subseteq Y$, there exists a character position $i \in \{1, \ldots, b\}$ such that some character appears an odd number of times in position i among the keys in Y'. A useful connection between this notion and tabulation-based hashing was shown by Thorup and Zhang [36], who proved that a set of keys is linearly independent if and only if simple tabulation hashing is fully random on these keys:

Lemma 2 (Simple tabulation on linearly independent keys)**.** *Given a set of keys $Y \subseteq \Sigma^b$ and a simple tabulation hash function $h : \Sigma^b \to \mathcal{R}$, the following are equivalent:*

(i) Y is linearly independent

(ii) h is fully random on Y (i.e., $h|_Y$ is distributed uniformly over $\mathcal{R}^Y$).[2]

To prove Theorem 1, we employ Lemma 2 with sets of derived keys. Namely, given a set of keys $X \subseteq \Sigma^c$, we consider the error event that the set $\widetilde{X} = \left\{\widetilde{h}(x) \mid x \in X\right\}$ of its derived keys is linearly dependent. We then show that this happens with probability at most $7|X|^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}$. If this doesn't happen, then the derived keys are linearly independent, and, by Lemma 2, we get that the tornado tabulation hash function $h = \widehat{h} \circ \widetilde{h}$ is fully random on X since it applies the simple tabulation hash function $\widehat{h}$ to the derived keys $\widetilde{X}$. We note that the general idea of creating linearly independent lookups to create fully-random hashing goes back at least to [33]. The point of this paper is to do it in a really efficient way.

Query and selected keys. Our main result, Theorem 5, is a more general version of Theorem 1. Specifically, while Theorem 1 holds for any fixed set of keys, it requires that $|X| \leqslant |\Sigma|/2$. For a fast implementation, we want $|\Sigma|$ to be small enough to fit in fast cache, e.g., $|\Sigma| = 2^8$, but in most applications, we want to hash a set S of keys that is much larger, e.g., $|S| \sim |\Sigma|^3$. Moreover, we might be interested in showing full randomness for subsets X of S that are not known in advance: consider, for instance, the set X of all the keys in S that hash near to $h(q)$ for some fixed key $q \in S$. In this case, Theorem 1 would not help us, since the set X depends on $h(q)$ hence on h.

To model this kind of scenario, we consider a set of *query keys* $Q \subseteq \Sigma^c$ and define a set of *selected keys* $X \subseteq \Sigma^c$. Whether a key x is selected or not depends only on x, its own hash value $h(x)$, and the hash values of the query keys $h|_Q$ (namely, conditioning on $h(x)$ and $h|_Q$ makes $x \in X$ and h independent). In Theorem 5, we will show that, if the selected keys are few enough, then $h|_X$ is fully random with high probability.

Formally, we have a selector function $f : \Sigma^c \times \mathcal{R} \times \mathcal{R}^Q \longrightarrow \{0,1\}$ and we define the set of selected keys as

$$X^{f,h} = \{x \in \Sigma^c \mid f(x, h(x), h|_Q) = 1\}\,.$$

We make the special requirement that f should always select all the query keys $q \in Q$, that is, $f(q,\cdot,\cdot) = 1$ regardless of the two last arguments. We then define

$$\mu^f := \sum_{x \in \Sigma^c} p_x^f \quad \text{with} \quad p_x^f := \max_{\varphi \in \mathcal{R}^Q} \Pr_{r \sim \mathcal{U}(\mathcal{R})} \left[f(x, r, \varphi) = 1\right]\,. \tag{2}$$

Here the maximum is taken among all possible assignments of hash values to query keys $\varphi : Q \to \mathcal{R}$ and r is distributed uniformly over $\mathcal{R}$. Trivially we have that

Observation 3. If $h : \Sigma^c \to \mathcal{R}$ is fully random then $\mathrm{E}[|X^{f,h}|] \leqslant \mu^f$.

When f and h are understood, we may omit these superscripts. It is important that X depends on both f and h while μ only depends on the selector f. We now have the following main technical theorem:

Theorem 4. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters and f as described above. If $\mu^f \leqslant \Sigma/2$ then the derived selected keys $\widetilde{h}(X^{f,h})$ are linearly dependent with probability at most $\mathsf{DependenceProb}(\mu^f, d, \Sigma)$, where*

$$\mathsf{DependenceProb}(\mu, d, \Sigma) := 7\mu^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}\,.$$

[2]In general, we employ the notation $h|_S$ to denote the function h restricted to the keys in some set S.

Note that Theorem 4 only bounds the probability of the error event. Similarly as in Theorem 1, we would like to then argue that, if the error event does not happen, we could apply Lemma 2 to claim that the final hash values via the simple tabulation function $\widehat{h}$ are fully random. The challenge, however, is the presence of an inherent dependency in how the keys are selected to begin with, namely that $h = \widehat{h} \circ \widetilde{h}$ is already used to select the keys in $X^{f,h}$. In other words, by the time we want to apply $\widehat{h}$ to the derived selected keys $\widetilde{h}(X^{f,h})$, we have already used some information about $\widehat{h}$ in selecting them in $X^{f,h}$.

Local uniformity. Nevertheless, there is a general type of selector functions for which we can employ Theorem 4 in conjunction with Lemma 2 to claim full randomness. Namely, we consider selector functions that partition the bit representation of the final hash values into *s selection bits* and *t free bits* so that $\mathcal{R} = \mathcal{R}_s \times \mathcal{R}_t = [2^s] \times [2^t]$. Given a key $x \in \Sigma^c$, we then denote by $h^{(s)}(x) \in \mathcal{R}_s$ and $h^{(t)}(x) \in \mathcal{R}_t$ the selection and free bits of $h(x)$ respectively. We then say that a selector function f is an *s-selector* if, for all $x \in \Sigma^c$, the output of $f(x, h(x), h|_Q)$ only depends on the selection bits of the hash function, i.e., $f(x, h(x), h|_Q) = f(x, h^{(s)}(x), h^{(s)}|_Q)$.

We now crucially exploit the fact that the output bits of a simple tabulation hash function are completely independent. Formally, we split the simple tabulation $\widehat{h}$ into two independent simple tabulation functions: $\widehat{h}^{(s)}$ producing the selection bits and $\widehat{h}^{(t)}$ producing the free bits. We then apply Theorem 4 to $h^{(s)} = \widehat{h}^{(s)} \circ \widetilde{h}$ to conclude that the set of selected derived keys $\widetilde{h}(X^{f,h^{(s)}})$ is linearly independent with high probability. Assuming this, we then apply Lemma 2 to conclude that $\widehat{h}^{(t)}$ is fully random on $\widetilde{h}(X^{f,\widehat{h}^{(s)}})$, hence that $h^{(t)} = \widehat{h}^{(t)} \circ \widetilde{h}$ is fully random on $X^{f,h^{(s)}}$.

Theorem 5. *Let $h : \Sigma^c \to \mathcal{R}$ be a tornado tabulation hash function with d derived characters and f be an s-selector as described above. If $\mu^f \leqslant \Sigma/2$, then $h^{(t)}$ is fully random on $X^{f,h^{(s)}}$ with probability at least*

$$1 - \mathsf{DependenceProb}(\mu^f, d, \Sigma) .$$

While the concept of an s-selector function might seem a bit cryptic, we note that it intuitively captures the notion of locality that linear probing and other applications depend on. Namely, the effect of the (high order[3]) selector bits is to specify a dyadic interval[4] of a given length such that all the keys with hash values falling in that interval are possibly selected (with this selection further depending, perhaps, on the query keys Q, or on other specific selector bits, leading to more refined dyadic intervals). Theorem 5 then says that the (low order) free bits of these selected keys will be fully-random with high probability. In other words, the distribution inside such a neighborhood is indistinguishable from what we would witness if we had used a fully-random hash function.

As mentioned earlier, the concept of local uniformity stems from [10], except that they did not consider query keys. Also, they didn't name the concept. They demonstrated its power in different streaming algorithms. For those applications, it is important that *the selection bits are not known to the algorithm.* They are only set in the analysis based on the concrete input to demonstrate good performance on this input. The problem in [10] is that their error probability bounds only apply when the alphabet is so large that the tables do not fit in fast cache. We will describe this issue closer in Section 1.3. In Section 1.4 we will sketch the use of local uniformity on linear probing where the locality is relative to a query key.

[3]Thinking about selector bits as higher order bits helps our intuition. However, they do not have to be higher-order bits necessarily. More generally, any representation of $\mathcal{R}$ as a product $\mathcal{R}_s \times \mathcal{R}_t$ would do the job.

[4]Recall that a dyadic interval is an interval of the form $[j2^i, (j+1)2^i)$, where i, j are integers.

Upper tail Chernoff bounds. Theorem 5 is only concerned with the distribution of the free bits of the selected keys, but to employ it in our applications, we often require that the number of selected keys is not much larger than μ^f with high probability (see Sections 1.4 and 1.6). We show that this size can be bounded from above with the usual Chernoff bound when the set of derived selected keys is linearly independent.

Lemma 6. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters, query keys Q and selector function f. Let $\mathcal{I}_{X^{f,h}}$ denote the event that the set of derived selected key $\widetilde{h}(X^{f,h})$ is linearly independent. Then, for any $\delta > 0$, the set $X^{f,h}$ of selected keys satisfies the following:*

$$\Pr\left[\left|X^{f,h}\right| \geqslant (1+\delta)\cdot\mu^f \wedge \mathcal{I}_{X^{f,h}}\right] \leqslant \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{\mu^f} .$$

For a nice direct application, consider hash tables with chaining, or throwing n keys into n bins using tornado hashing. We can select a given bin, or the bin of a given query key. In either case we have $\mu^f = 1$ and then Lemma 6 together with Theorem 4 says that the probability of getting k keys is bounded by

$$e^{k-1}/k^k + 7(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}. \tag{3}$$

If k is not too large, the first term dominates.

Lower bound. Finally, we show that our upper bound for the error probability in Theorem 4 is tight within a constant factor.

Theorem 7. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters. There exists a selector function f with $\mu^f \leqslant \Sigma/2$ such that the derived selected keys $\widetilde{h}(X^{f,h})$ are linearly dependent with probability at least $\Omega((3/|\Sigma|)^{d-2})$.*

Full randomness for larger sets of selected keys. As mentioned earlier, for a fast implementation of tabulation hashing, we pick the alphabet Σ small enough for the tables to fit in fast cache. A common choice is 8-bit characters. However, we only get full randomness for (selected) sets of (expected) size at most $|\Sigma|/2$ (c.f. Theorems 5 and 1).

To handle larger sets, we prove that it suffices to only increase the alphabet of the last two derived characters; meaning that only two lookups have to use larger tables. This is close to best possible in that we trivially need at least one lookup in a table at least as big as the set we want full randomness over. More precisely, if we use alphabet Ψ for the last two derived characters, then our size bound increases to $|\Psi|/2$. The error probability bound of Theorems 5 becomes

$$14(\mu^f)^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} + 1/2^{|\Sigma|/2} .$$

With the above mix of alphabets, we have tornado hashing running in fast cache except for the last two lookups that could dominate the overall cost, both in time and space. Because they dominate, we will consider a slight variant, where we do not store derived characters in any of the two large tables. Essentially this means that we change the definition of the last derived character from $\widetilde{x}_{c+d} = \widetilde{h}_d(\widetilde{x}_1 \cdots \widetilde{x}_{c+d-1})$ to $\widetilde{x}_{c+d} = \widetilde{h}_d(\widetilde{x}_1 \cdots \widetilde{x}_{c+d-2})$. This is going to cost us a factor two in the error probability, but in our implementation, we will now have $c+d-2$ lookups in tables $\Sigma \to \Sigma^{d-1} \times \Psi^2 \times R$ (where the values are represented as a single w-bit numbers), and 2 lookups in tables $\Psi \to R$. We shall refer to this scheme as *tornado-mix*. Corresponding to Theorem 5, we get

Theorem 8. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado-mix tabulation hash function with d derived characters, the last two from Ψ, and an s-selector function f. If $\mu = \mu^f \leqslant |\Psi|/2$ then $h^{(t)}$ is fully random on $X^{f,h}$ with probability at least*

$$1 - 14\mu^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} - 1/2^{|\Sigma|/2}\ .$$

An interesting case is when we want linear space uniform hashing for a fixed set X, in which case $\mu = |X|$ above. This leads to a much better implementation of the uniform hashing fo Pagh and Pagh [28]. The main part of their paper was to show a linear space implementation, and this would suffice for all but the most succinct space algorithms. They used highly independent hashing [33, 35] as a subroutine, but this subroutine alone is orders of magnitude slower than our simple implementation (see, e.g., [1]). They combined this with a general reduction from succinct space to linear space, for which we now have a really efficient construction.

1.3 Techniques and relation to mixed tabulation

In spirit, our results are very related to the results on mixed tabulation [10]. For now, we only consider the case of a single alphabet Σ. Indeed, tornado and mixed tabulation are very similar to implement. Both deal with c-character keys from some alphabet Σ, produce a derived key with $c+d$ characters, and then apply a top simple tabulation to the resulting derived keys. Both schemes can be implemented with $c+d$ lookups. The difference is in how the two schemes compute the derived keys. For ease of presentation, let $\widetilde{h}_i : \Sigma^* \to \Sigma$, that is, $\widetilde{h}_i$ adjusts to the number of characters in the input. Now for mixed tabulation, we define the derived key $\widetilde{x}_1 \cdots \widetilde{x}_{c+d}$ by

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i \leqslant c \\ \widetilde{h}_{i-c}\left(\widetilde{x}_1 \ldots \widetilde{x}_c\right) & \text{if } i > c. \end{cases}$$

The analysis from [10] did not consider query keys, but ignoring this issue, their analysis works in the limit $|\Sigma| \to \infty$. For example, the analysis from [10] requires that [5]

$$(\log|\Sigma|)^c \leqslant |\Sigma|/2.$$

This is true for $c = O(1)$ and $|\Sigma| \to \infty$, but simply false for realistic parameters. Assuming the above condition to be satisfied, if we consider scenarios with non-constant c and d, the error probability from [10] becomes

$$(O(cd)^c/|\Sigma|)^{\lfloor d/2 \rfloor - 1} + 1/2^{\Omega(|\Sigma|)}.$$

Now, even if we replace $O(cd)$ with cd, the error probability is not below 1 even with 16-bit characters and $c = 4$. In contrast, practical tabulation schemes normally use 8-bit characters for efficiency, and our explicit bound of $7\mu^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}$ from Theorem 4 works fine even in this case, implying that our theory actually applies to practice.

The reason that mixed tabulation has the problematic exponential dependency on c is that for a set of linearly dependent keys, it uses a clever encoding of each of the c characters in some of the keys. With tornado, the only encoding we use is that if we have a zero set of keys, then each key is the xor of the other keys in the zero set, and this is independent of c. [6]

[5] While using their Lemma 3, if t goes up to s in case D.

[6] A set is linearly dependent if it contains a subset that is a zero set. See Section 2 for the precise definition.

To describe the advantage of tornado tabulation over mixed tabulation, it is easier to first compare with what we call *simple tornado* where the derived key $\widetilde{x}_1 \cdots \widetilde{x}_{c+d}$ is defined by

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i \leqslant c \\ \widetilde{h}_{i-c}\left(\widetilde{x}_1 \ldots \widetilde{x}_{i-1}\right) & \text{if } i > c. \end{cases}$$

The implementation difference is that with simple tornado, each derived character uses all preceding characters while mixed tabulation uses only the original characters. This difference gives tornado a big advantage when it comes to breaking up linear dependence in the input keys. Recall that a set $Y \subset \Sigma^{c+d}$ of derived keys is linearly independent if and only if, for every subset $Y' \subseteq Y$, there exists a character position $i \in \{1, \ldots, c+d\}$ such that some character appears an odd number of times in position i among the keys in Y'. Intuitively, the strategy is to argue that whatever linear dependence exists in the input key set to begin with (essentially, in the first c characters of the derived keys) will, whp, be broken down by their d derived characters (and hence, disappear in the derived keys). In this context, the way we compute the derived characters becomes crucial: in mixed tabulation, each derived character is computed independently of the other derived characters. Thus, whether a derived character breaks a dependency or not is independent of what other derived characters do. In contrast, we make the derived characters in tornado tabulation depend on all previously computed derived characters such that, if we know that some derived character does not break a dependency, then we also know that none of its previously computed derived characters have broken it either. Or, in other words, each successive tornado-derived character benefits from the dependencies already broken by previously computed derived characters, i.e., the benefits compound each time we compute a new derived character.

This structural dependence between tornado-derived characters turns out to be very powerful in breaking linear dependencies among the input keys and indeed, leads to a much cleaner analysis. The most important benefit, however, is that tornado tabulation has a much lower error probability. For simple tornado, we get an error probability of

$$7(\mu^f)^3(3/|\Sigma|)^d + 1/2^{|\Sigma|/2} ,$$

which essentially gains a factor $(3/|\Sigma|)$ for each derived character. It turns out that we gain an extra factor $(3/|\Sigma|)$ if we twist the last original character as we did in the original tornado definition (1), and then we get the bound from Theorem 5, the point being that this twisting does not increase the number of lookups.

One might wonder if twisting more characters would help further, that is, setting $\widetilde{x}_i = x_i \oplus \widetilde{h}_i(\widetilde{x}_1 \cdots x_{i-1})$ for $i = 2, \ldots, c$, but it doesn't. The point is that the bad key set from our lower bound in Theorem 7 is of the form $[0]^{c-2} \times [2] \times A$ for some $A \subseteq \Sigma$, and then it is only the last character where twisting makes a difference.

As a last point, recall tornado-mix which was designed to deal with larger sets. There it only costs us a factor 2 when we let the last derived character $\widetilde{x}_{c+d}$ depend only on $\widetilde{x}_1 \ldots \widetilde{x}_{c+d-2}$ and not on $\widetilde{x}_{c+d-1}$. This is essentially like switching to mixed tabulation on the last two derived keys, hence the name *tornado-mix*. This only works for the last two derived characters that can play a symmetric role.

The exponential dependence on c. We note that having an exponential dependence on c is symptomatic for almost all prior work on tabulation hashing, starting from the original work in [31]. Above, we discussed how tornado tabulation hashing avoided such exponential dependence on

c in the error probability from mixed tabulation. The dependence on c was particularly destructive for mixed tabulation because it pushed the error probability above 1 for the relevant parameters.

Concentration bounds. Tornado hashing inherits the strongest concentration bounds known for mixed tabulation [22]. The reason is that they only require one derived character and tornado tabulation can be seen as applying mixed tabulation with one character to a derived tornado key with one less character. Unlike our Lemma 6, which essentially only applies for expected values $\mu \leqslant |\Sigma|/2$, the concentration bounds we get from [22] work for unbounded μ and for both the upper and lower tail. They fall exponentially in μ/K_c where K_c is exponential in c, but this still yields strong bounds when μ is large. Inheriting the strongest concentration bound for mixed tabulation implies that tornado tabulation can replace mixed tabulation in all the applications from [10] while improving on the issue of local uniformity.

1.4 The power of locally uniform hashing

We now describe, on a high level, how the notion of locally uniform hashing captures a type of randomness that is sufficient for many applications to work almost as if they employed full randomness. For the discussion, we will assume the parameters from Theorem 8 with tornado-mix. The main observation is that, for many algorithms, the performance measure (i.e., its distribution) depends, whp, only on the behavior of keys that hash inside a local neighborhood defined via some selected bits of the hash value (the selection may depend both on the algorithm and on the concrete input since the selection is only used for analysis). Moreover, the set of keys $X^{f,h}$ that land in that selected neighborhood has expected size $\leqslant |\Psi|/2$. For any such neighborhood, our results imply that the keys in $X^{f,h}$ have fully random free bits whp.

To understand the role of the free bits, it is helpful to think of hashing a key x as a two-stage process: the selector bits of $h(x)$ tell us whether x hashes in the desired neighborhood or not, while the remaining free bits of $h(x)$ determine how x hashes once it is *inside* the neighborhood. This suggests a general coupling, where we let both fully-random hashing and tornado tabulation hashing first choose the select bits and then the free bits. Since both are fully random on the free bits (for us with high probability), the only difference is in the selection, but here concentration bounds imply that we select almost the same number of keys as fully-random hashing.

In [10], this approach was demonstrated for the HyperLogLog algorithm of Flajolet, Fusy, Gandouet, Meunier [18] for counting distinct elements in data streams, and the One-Permutation Hashing of Li, Owen, and Zhang [26] used for fast set similarity in large scale machine learning. In [10] they implemented the local uniformity with mixed tabulation, but with tornado hashing we get a realistic implementation. This also includes later application of mixed tabulation, e.g., the dynamic load balancing from [3]. Below, as a new example, we illustrate how local uniformity makes classic linear probing perform almost as if fully random hashing was used.

We briefly recall the basic version of linear probing. We employ an array T of length m and hash keys to array entries using a tornado tabulation hash function $h : \Sigma^c \to [m]$. Thus, in Theorem 8, we have $\mathcal{R} = [m] = [2^r]$. Upon insertion of a key x, we first check if entry $T[h(q)]$ is empty. If it is, we insert the key in $T[h(q)]$. Otherwise, we scan positions in T sequentially starting with $T[h(q) + 1]$ until we find an empty position and store x in this next available free entry. To search for x, we inspect array entries sequentially starting with $T[h(q)]$ until we find x or we find an empty array entry, concluding that the key is not in the array. The general concept that dominates the performance of linear probing is the *run* of q, which is defined as the longest interval $I_q \subseteq [m]$ of full (consecutive) array entries that $h(q)$ is part of. The run affects the probe length (the length of the interval from $h(q)$ till the end of I_q) and other monotone measures such as insertion and deletion time (i.e., monotonically increasing in the number of keys). Thus, any corresponding analysis depends only on the set X_q of keys that fall in the interval I_q, and it is there that we are interested in showing full randomness.

The first step is to argue that the behavior of I_q is local in nature, in that it is affected only by the keys that hash in a fixed length interval around $h(q)$. To that end, we show that, whp, the length of the run is no bigger than a specific $\Delta = 2^\ell$. This Δ implies locality: consider the interval $J_q \subseteq [m]$, centered at $h(q)$, which extends Δ entries in each direction, i.e., $J_q = \{j \in [m] \mid |j - h(q)| \leqslant \Delta\}$. Then, whp, any run that starts outside J_q is guaranteed to end by the time we start the run of q. In other words, whp, the behavior of I_q depends only on the set X_q of keys that hash inside the fixed length neighborhood J_q (and specifically where in J_q the keys hash). A similar argument can be made for other variants of linear probing, such as linear probing with tombstones [6], where the run/neighborhood must also take into account the tombstones that have been inserted.

At this point, it is worthwhile pointing out that the set X_q is no longer a fixed set of keys but rather a random variable that depends on the realization of $h(q)$. We cannot just apply Theorem 1. Nevertheless, in the second step, we argue that X_q can be captured by our notion of selector functions. For this, we cover J_q with three dyadic intervals, each of length Δ: one including $h(q)$ and the corresponding ones to the left and to the right. Since each interval is dyadic, it is determined only by the leftmost $r - \ell$ bits of the hash value: for example, an element $x \in \Sigma^c$ hashes into the same dyadic interval as q iff the leftmost $r - \ell$ bits of $h(x)$ match those of $h(q)$. We can then design a selector function that returns a 1 if and only if x is in the input set and the leftmost $r - \ell$ bits of $h(x)$ (its selector bits) hash it into any of the three specific dyadic intervals we care about. Such a selector function is guaranteed to select all the keys in X_q. By setting Δ appropriately, we get that the expected size of the selected set is at most $\Sigma/2$, and can thus apply Theorem 5. We get that inside the intervals, keys from $X^{f,h}$ hash (based on their free bits) in a fully random fashion.

Finally, what is left to argue is that the number of keys hashing inside each of the three dyadic intervals is not much bigger than what we would get if we used a fully-random hash function for the entire set (one in which the selector bits are also fully random). For this, we employ Lemma 6, and can conclude that we perform almost as well with fully-random hashing. In particular, from Knuth's [25] analysis of linear probing with fully-random hashing, we conclude that with load $(1 - \varepsilon)$, the expected probe length with tornado hashing is $(1 + o(1))(1 + 1/\varepsilon^2)/2$.

The above type of analysis could be applied to other variants of linear probing, e.g., including lazy deletions and tombstones as in the recent work of Bender, Kuszmaul, and Kuszmaul [6]. We would need to argue that the run, including tombstones, remains within the selected neighborhood and that we have concentration both on live keys and tombstones. However, since the analysis from [6] is already based on simple hash functions, we would not get new bounds from knowing that tornado hashing performs almost as well as fully-random hashing.

1.5 Relation to high independence

The k-independence approach to hashing [37] is to construct hash functions that map every set of k keys independently and uniformly at random. This is much stronger than getting fully random hashing for a given set and, not surprisingly, the results for highly independent hashing [9, 33, 35] are weaker. The strongest high independence result from [9] says that we get $|\Sigma|^{1-\varepsilon}$-independent hashing in $O((c/\varepsilon)\log(c/\varepsilon))$ time, but the independence is much less than our $|\Sigma|/2$. Experiments from [1] found that even the simpler non-recursive highly independent hashing from [35] was more than an order of magnitude slower than mixed tabulation which is similar to our tornado tabulation.

We also note that the whole idea of using derived characters to get linear independence between keys came from attempts to get higher k-independence Indeed, [15, 23, 36] derive their characters deterministically, guaranteeing that we get k-independence. The construction for $k = 5$ is efficient but for larger k, the best deterministic construction is that from [36] using $d = (k-1)(c-1)+1$ derived characters, which for larger k is much worse than the randomized constructions.

1.6 Relation to the splitting trick and to succinct uniform hashing

We will now describe how tornado-mix provides a much more efficient implementation of the succinct uniform hashing by Pagh and Pagh [28] and the splitting trick of Dietzfelbinger and Rink [13]. This further illustrates how our work provides a component of wide applicability within hashing.

Succinct uniform hashing. The main contribution in [28] is to obtain uniform hashing in linear space. The succinct construction is then obtained through a general reduction from the linear-space case. They achieved uniform hashing in linear space using the highly independent hashing of Siegel [33] as a subroutine in combination with other ideas. Now, thanks to Theorem 8, we know that tornado-mix offers an extremely simple and efficient alternative to implement that. Indeed, if n is the size of the set we want linear space uniform hashing on, then setting $|\Psi|$ to the power of two just above $2n$ is sufficient to ensure the degree of independence that we need. Moreover, we can set $|\Sigma| \sim \sqrt{\Psi}$ and c so that we obtain (i) $c|\Sigma| = o(|\Psi|)$ and (ii) setting $d = 2c+1$ the probability bound in Theorem 8 becomes $1 - O(1/u)$ where u is the size of the universe. The two previous conditions ensure respectively that tornado-mix uses linear space and that it is, whp, fully random.

The splitting trick. We now explain how our tornado-mix tabulation hashing provides a very simple and efficient implementation of the splitting trick of Dietzfelbinger and Rink [13] which is a popular method for simulating full-randomness in the context of data structures, most notably various dictionary constructions [14, 16, 19, 5]. This is an especially relevant application because it usually requires hashing keys into a range of size $\Theta(n)$ and, in this context, employing the uniform hashing of Pagh and Pagh [28] would be too costly, i.e., (compared to the size of the overall data structure). This trick was also used to obtain a simpler and exponentially faster implementation of succinct uniform hashing [28]. We note that the splitting idea had also been used earlier works of Dietzfelbinger and Meyer auf der Heide [12] and Dietzfelbinger and Woelfel [15].

The idea is to first split the input set S of size n into $n^{1-\delta}$ subsets $S_1, \ldots, S_m$, for some $\delta \in (0,1)$. The splitting is done through a hash function $h_1 : \Sigma^c \to [n^{1-\delta}]$ such that S_i is defined as all the keys in S that hash to the same value, i.e., $S_i = \{x \in S \mid h_1(x) = i\}$. In many applications, we want the splitting to be balanced, that is, we need a joint upper bound s on all set sizes with s close to the expected size n^{δ}. On top of this, we need a shared hash function $h_2 : \Sigma^c \to \mathcal{R}$ which with high probability is fully random on each set S_i and we ensure this with a hash function that w.h.p. is fully random on any set of size at most s. The problem is then solved separately for each subset (e.g., building $n^{1-\delta}$ dictionaries, each responsible for just one subset S_i).

The above splitting trick can be easily done with tornado-mix hashing from Theorem 8. The dominant cost for larger s is two lookups in tables of size at most $4s$. We let h_1 be the select bits (with the strong concentration from Lemma 6) and h_2 as the remaining free bits. Getting both for the price of one is nice, but the most important thing is that we get an efficient implementation of the shared hash function h_2. Specifically, we compare this with how h_1 and h_2 were implemented in [13] to get uniform hashing. For the splitter h_1, instead of using Siegel's [33] highly independent hashing, [13, §3.1] uses that if h_1 has sufficiently high, but constant, independence, then it offers good concentration (though not as good as ours from Lemma 6).

The bigger issue is in the implementation of the shared h_2 from [13, §3.2]. For this, they first employ a hash function $f : [s] \to [s^{1+\varepsilon}]$ such that, whp, f has at most $k = O(1)$ keys that get the same hash value. For small ε and k, this forces a high independence of f. Next, for every $i \in [s^{1+\varepsilon}]$, they use a k-independent hash function $g_i : \Sigma^c \to \mathcal{R}$, and finally, h_2 is implemented as $g_{f(x)}(x)$. The space to implement h_2 is dominated by the $O(s^{1+\varepsilon})$ space to store the $s^{1+\varepsilon}$ k-independent hash functions g_i, and time-wise, we have to run several sufficiently independent hash functions. This should be compared with our tornado-mix that only uses $O(s)$ space and runs in time corresponding to a few multiplications (in tests with 32-bit keys).

1.7 Paper Organization

The remainder of our paper is structured as follows. In Section 2, we introduce some notation, a more general notion of generalized keys and Lemma 9 (the corresponding version of Lemma 2 for them). Our main technical result, Theorem 4, is proved in three steps: we first define obstructions, the main combinatorial objects we study, in Section 3. In Section 4 we present a simplified analysis of Theorem 4 that achieves a weaker error probability. We then show in Section 5 a tighter analysis that finally achieves the desired bound. The Chernoff bounds for the upper tail are proved in Section 6. The details for the linear probing analysis can be found in Section 7. Finally, the lower bound from Theorem 7 is proved in Section 8.

2 Preliminaries

Notation. We use the notation $n!! = n(n-2)\cdots$. More precisely, $n!! = 1$ for $n \in \{0,1\}$, while $n!! = n(n-2)!!$ for $n > 1$. For odd n, this is exactly the number of perfect matchings of $n+1$ nodes. We use the notations

$$n^{\underline{k}} = n(n-1)\cdots(n+1-k) = n!/(n-k)!$$
$$n^{\underline{\underline{k}}} = n(n-2)\cdots(n+2(1-k)) = n!!/(n-2k)!$$

Here $n^{\underline{\underline{k}}}$ appears to be non-standard, though it will be very useful in this paper in connection with something we will call greedy matchings. We note that

$$n^{\underline{\underline{k}}} \leqslant n^{\underline{k}} \leqslant n^k.$$

Position characters and generalized keys. We employ a simple generalization of keys going back to Pătraşcu and Thorup [31]. Namely, a *position character* is an element of $\{1 \dots b\} \times \Sigma$, e.g., where $b = c$ or $c + d$. Under this definition a key $x \in \Sigma^b$ can be viewed as a set of b position characters $(1, x_1) \dots (b, x_b)$, and, in general, we consider *generalized keys* that may be arbitrary subsets of $\{1 \dots b\} \times \Sigma$. A natural example of a generalized key is the symmetric difference $x \triangle y$ of two (regular) keys. We then have that

$$h(x) = h(y) \iff h(x \triangle y) = 0.$$

These symmetric differences will play an important role in our constructions, and we shall refer to $x \triangle y$ as a *diff-key.* A diff-key is thus a generalized key where we have zero or two characters in each position. For a generalized key x, we can then define

$$x[i] = \{(i,a) \in x\}, \quad x[< i] = \{(j,a) \in x \mid j < i\} \quad \text{and} \quad x[\leqslant i] = \{(j,a) \in x \mid j \leqslant i\}.$$

For example, if we have a regular key $x = x_1, \ldots, x_c$, then $x[i] = x_i$ and $x[\leqslant i] = x_1 \ldots, x_i$. Also note that $(x \triangle y)[\leqslant i] = x[\leqslant i] \triangle y[\leqslant i]$. We shall also apply this indexing to sets X of generalized keys by applying it to each key individually, e.g.,

$$X[< i] = \{x[< i] \mid x \in X\}.$$

Generalized keys and linear independence. A generalized key x can also be interpreted as a $(|\Sigma| \cdot b)$-dimensional vector over $\mathbb{F}_2$, where the only entries set to 1 are those indexed by position characters in x. The generalized key domain is denoted by $\mathbb{F}_2^{\{1\ldots b\} \times \Sigma}$. Now, if we have a simple tabulation hash function $h : \Sigma^b \to \mathcal{R}$ using character tables $T_1, \ldots, T_b$, then h can be lifted to hash any generalized key $x \in \mathbb{F}_2^{\{1\ldots b\} \times \Sigma}$ by

$$h(x) = \bigoplus_{(i,a) \in x} T_i[a].$$

Thus h provides a mapping $\mathbb{F}_2^{\{1\ldots b\} \times \Sigma}$ to $\mathcal{R}$.

As for (regular) keys, for a set Y of generalized keys, we can then define $\triangle Y$ to be the set of position characters that appear an odd number of times across the keys in Y. We then say that Y is a *zero-set* if $\triangle Y = \varnothing$. We also say that Y is *linearly dependent* if it contains a subset which is a zero-set, and *linearly independent* otherwise. It is apparent then that a set of generalized keys is linearly independent if and only if the set of their vector representations is linearly independent. Indeed, the proof of Thorup and Zhang [36] for Lemma 9 works quite directly in this generality. Thus we have

Lemma 9 (Simple tabulation on linearly independent generalised keys)**.** *Given a set of generalized keys $Y \subseteq \left(\mathbb{F}_2^{\{1\ldots c\} \times \Sigma}\right)$ and a simple tabulation hash function $h : \Sigma^c \to \mathcal{R}$, the following are equivalent:*

(i) Y is linearly independent

(ii) h is fully random on Y (i.e., $h|_Y$ is distributed uniformly over $\mathcal{R}^Y$).

3 Obstructions with simple tornado tabulation

We prove Theorem 4 by first considering a simpler version of tornado tabulation hashing, which we call *simple tornado hashing*, where we do not change the last character of the (original) key. Formally, for a key $x = x_1 \cdots x_c$, its corresponding derived key $\widetilde{x} = \widetilde{x}_1 \ldots \widetilde{x}_{c+d}$ is computed as

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i = 1, \ldots, c \\ \widetilde{h}_{i-c}(\widetilde{x}_1 \ldots \widetilde{x}_{i-1}) & \text{otherwise.} \end{cases}$$

Note that, in the original tornado hashing, we had $\widetilde{x}_c = x_c \oplus \widetilde{h}_0(\widetilde{x}_1 \ldots \widetilde{x}_{c-1})$. Removing this extra step is thus equivalent to fixing $\widetilde{h}_0(\cdot) = 0$. While this step comes at almost no cost in the code, it allows us to gain a factor of $3/|\Sigma|$ in the overall error probability. See Section 5.3 for details. For the simple tornado hashing, we will prove a slightly weaker probability bound.

For ease of notation, for every key x, we use $\widetilde{x}$ to denote the corresponding derived key $\widetilde{h}(x)$; and likewise for any set of keys. We also define $X = X^{f,h}$ to be the set of selected keys and $\widetilde{X} = \widetilde{h}(X)$. We want to argue that the derived selected keys $\widetilde{X}$ are linearly independent with high probability. To prove this, we assume that $\widetilde{X}$ is linearly dependent and hence, contains some zero-set $\widetilde{Z}$. From $\widetilde{Z}$ we construct a certain type of "obstruction" that we show is unlikely to occur.

3.1 Levels and matchings

We first define some necessary concepts. We use the notion of *level i* to refer to position $c+i$ in the derived keys. Let $M \subseteq \binom{|\Sigma|^c}{2}$ be a (partial) matching on the keys Σ^c.[7] We say that M is an *i-matching* if for all $\{x, y\} \in M$, it holds that $\widetilde{x}[c+i] = \widetilde{y}[c+i]$, namely if every pair of keys in M matches on level i. Our obstruction will, among other things, contain an i-matching M_i for each level i.

Recall that a diff-key $x \triangle y$ is the symmetric difference of two keys x and y in terms of their position characters. We then say that M is an *i-zero, i-dependent, or i-independent* matching if

$$\mathrm{DiffKeys}(M, i) = \{(\widetilde{x} \triangle \widetilde{y})[\leqslant c+i] \mid \{x, y\} \in M\}$$

is a zero-set, linearly dependent, or linearly independent, respectively. In other words, for each pair $\{x, y\}$ in the matching, we consider the diff-key corresponding to the first $c+i$ characters of their derived keys. We then ask if this set of diff-keys now forms a zero-set or contains one as a subset, by looking at their (collective) symmetric difference. We employ this notion to derive the probability that the function $\widetilde{h}$ satisfies a certain matching as such:

Lemma 10. *Let M be a partial matching on Σ^c. Conditioning on M being $(i-1)$-independent, M is an i-matching with probability $1/|\Sigma|^{|M|}$.*

Proof. First, we notice that the event "M is $(i-1)$-independent" only depends on $\widetilde{h}_j$ for $j < i$. Then, by Lemma 9, when we apply the simple tabulation hash function $\widetilde{h}_i : \Sigma^{c+i} \to \Sigma$ to the linearly independent generalized key set $\mathrm{DiffKeys}(M, i-1)$, the resulting hash values $\widetilde{h}(z)[c+i]$ for $z \in \mathrm{DiffKeys}(M, i-1)$ are independent and uniform over Σ. Hence, so are the resulting derived characters $\widetilde{h}(z)[c+i]$ for $z \in \mathrm{DiffKeys}(M, i)$. The probability that they are all 0 is therefore $1/|\Sigma|^{|M|}$. ☐

Similarly, as for matchings, we say that a set of keys $Z \subseteq \Sigma^c$ is *i-zero*, *i-dependent, or i-independent* if $\widetilde{Z}[\leqslant c+i]$ is a zero-set, linearly dependent, or linearly independent, respectively. We note the following relations:

Observation 11. Let M be a partial matching on Σ^c and $Z = \bigcup M$. Then M is an i-zero matching iff Z is an i-zero set. Furthermore, if M is i-dependent then Z is also i-dependent (but not vice versa).

[7]Here, we mean the graph-theoretic definition of a matching as a set of edges with disjoint endpoints. In our case, the vertices of the graph are keys in Σ^c, and the edges of the matching are represented as $\{x, y\} \in M$.

We will also make use of the following observation repeatedly:

Observation 12. If Z is an i-zero set, then there is a perfect j-matching on Z for every level $j \leqslant i$.

3.2 Constructing an obstruction

In this section, we show that whenever the set of selected derived keys $\widetilde{X}$ is linearly dependent, it gives rise to a certain *obstruction.* We now show how to construct such an obstruction.

Since $\widetilde{X}$ is linearly dependent, there must be some d-zero set $Z \subseteq X$. We are going to pick a d-zero set that (1) minimizes the number of elements contained that are not in the query set Q and, subject to (1), (2) minimizes the number of elements from Q contained. In particular, Z is contained in Q if $\widetilde{Q}$ is not linearly independent.

If Z is not contained in Q, we let x^* be any element from $Z \backslash Q$. Then $Q \cup Z \backslash \{x^*\}$ must be linearly independent since any strict subset Z' would contradict (1). If Z is contained in Q, then we let x^* be an arbitrary element of Z.

The top two levels. By Observation 12, we have a perfect d-matching M_d^* and a perfect $(d-1)$-matching M_{d-1}^* on Z (we also have perfect matchings on other levels, but they will be treated later). These two perfect matchings partition Z into alternating cycles.

We will now traverse these cycles in an arbitrary order except that we traverse the cycle containing x^* last. For all but the last cycle, we start in an arbitrary vertex and its cycle starting with the edge from M_{d-1}^*. When we get to the last cycle, we start at the M_d^* neighbor of x^* and follow the cycle from the M_{d-1}^* neighbor of this key. This ensures that x^* will be the very last vertex visited. The result is a traversal sequence $x_1, \ldots, x_{|Z|}$ of the vertices in Z ending in $x_{|Z|} = x^*$. Note that M_{d-1}^* contains the pairs $\{x_1, x_2\}, \{x_3, x_4\}, \ldots$. For M_d^* it is a bit more complicated, since its pairs may be used to complete a cycle (and hence are not visible in the traversal).

We now define $W = \{x_1 \ldots x_w\}$ to be the shortest prefix of $x_1 \ldots x_{|Z|}$ such that M_{d-1}^* restricted to the keys in W is a $(d-1)$-dependent matching, i.e., we go through the pairs $\{x_1, x_2\}, \{x_3, x_4\}, \ldots$ until the set of their diff-keys (up to level $d-1$) contains a zero-set. Note that such a $W \subseteq Z$ always exists because M_{d-1}^* itself is a $(d-1)$-zero matching. Also note that $W \backslash \{x_w\} \subseteq Z \backslash \{x^*\}$. Let $e_{d-1} := \{x_{w-1}, x_w\}$ be the last pair in the prefix, and as $e_{d-1} \in M_{d-1}^*$, we get that w is even. We then define M_{d-1} to be the restriction of M_{d-1}^* to the keys in W and M_d to be the restriction of M_d^* to the keys in $W \backslash \{x_w\}$. Note that M_{d-1} is a perfect matching on W while M_d is a maximal matching on $W \backslash \{x_w\}$. Since M_{d-1} is $(d-1)$-dependent, we can define a submatching $L_{d-1} \subseteq M_{d-1}$ such that L_{d-1} is a $(d-1)$-zero matching (this corresponds exactly to the subset of DiffKeys$(M_{d-1}, d-1)$ that is a zero-set). By construction, $e_{d-1} \in L_{d-1}$. Finally, we set $Z_{d-1} = \bigcup L_{d-1}$ and notice that Z_{d-1} is an $(d-1)$-zero key set (by Observation 11).

A special total order. We now define a special new total order $\preceq$ on Σ^c that we use in order to index the keys in W and describe matchings on levels $< d-1$. Here x_w has a special role and we place it in a special position; namely at the end. More precisely, we have the natural $\leqslant$-order on Σ^c, i.e, in which keys are viewed as numbers $< |\Sigma|^c$. We define $\preceq$ to be exactly as $\leqslant$ except that we set x_w to be the largest element. Moreover, we extend the total order $\preceq$ to disjoint edges in a matching M: given $\{x_1, x_2\}, \{y_1, y_2\} \in M$, we define $\{x_1, x_2\} \preceq \{y_1, y_2\}$ if and only if $\min x_i \prec \min_i y_i$.

Lower levels. Now for $i = d-2 \dots 0$, we do the following: from level $i+1$, we have an $(i+1)$-zero set Z_{i+1}. By Observation 12, there exists a perfect i-matching M_i^* over Z_{i+1}. Notice that M_i^* is an i-zero matching. We define M_i as the shortest prefix (according to $\preceq$) of M_i^* that is i-dependent. Denote by e_i the $\preceq$-maximum edge in M_i. Define the submatching $L_i \subseteq M_i$ such that L_i is an i-zero set. By construction, $e_i \in L_i$. Finally, we set $Z_i = \bigcup L_i$ and notice that Z_i is an i-zero set.

3.3 Characterizing an obstruction

Our main proof strategy will be to show that an obstruction is unlikely to happen, implying that our selected derived keys $\widetilde{X}$ are unlikely to be linearly dependent. To get to that point, we first characterize such an obstruction in a way that is independent of how it was derived in Section 3.2. With this classification in hand, we will then be able to make a union bound over all possible obstructions.

Corresponding to the two top levels, our obstruction consists of the following objects:

- A set of keys $W \subseteq \Sigma^c$ of some size $w = |W|$.
- A special key $x_w \in W$, that we put last when we define $\preceq$.
- A maximal matching M_d on $W \backslash \{x_w\}$.
- A perfect matching M_{d-1} on W, where e_{d-1} is the only edge in M_{d-1} incident to x_w.
- A submatching $L_{d-1} \subseteq M_{d-1}$, which includes e_{d-1}, and its support $Z_{d-1} = \bigcup L_{d-1}$.

Note that the above objects do not include the whole selected set $X^{f,h}$, the d-zero set Z, or the perfect matchings M_{d-1}^*, M_d^* that we used in our construction of the obstruction. Also, note that thus far, the objects have not mentioned any relation to the hash function h.

To describe the lower levels, we need to define greedy matchings. We say a matching M on a set Y is *greedy* with respect to the total order $\preceq$ on Y if either: (i) $M = \varnothing$ or; (ii) the key $\min_{\preceq} Y$ is incident to some $e \in M$ and $M \backslash \{e\}$ is greedy on $Y \backslash e$. Assuming that $|Y|$ is even, we note that M is greedy if and only if it is a prefix of some $\preceq$-ordered perfect matching M^* on Y.

For the lower levels $i = d-2, \dots, 1$, we then have the following objects:

- A greedy matching M_i on Z_{i+1}. Denote with e_i the $\preceq$-maximum edge in M_i.
- A submatching $L_i \subseteq M_i$ which includes e_i, and its support $Z_i = \bigcup L_i$.

Note that the above objects describe all possible obstructions that we might construct. Moreover, any obstruction can be uniquely described as the tuple of objects

$$(W, x_w, M_d, M_{d-1}, e_{d-1}, L_{d-1}, Z_{d-1}, \dots M_1, e_1, L_1, Z_1) .$$

3.4 Confirming an obstruction

In order for a given obstruction to actually occur, the tornado tabulation hash function $h = \widehat{h} \circ \widetilde{h}$ must satisfy the following conditions with respect to it:

- The keys in W are all selected, that is, $W \subseteq X^{f,h}$.

- Either $W \subseteq Q$ or $Q \cup W\backslash\{x_w\}$ is d-independent.
- For $i = 1, \dots, d$, M_i is an i-matching.
- M_d is $(d-1)$-independent.

For $i = 1, \dots, d-1$,

- $M_i \backslash \{e_i\}$ is $(i-1)$-independent.
- The submatching L_i is i-zero (implying that $Z_i = \bigcup L_i$ is an i-zero set).

When a hash function h satisfies the above conditions, we say that it *confirms* an obstruction. We now need to argue two things. We need (1) to verify that our construction of an obstruction from Section 3.2 satisfies these conditions on h, and (2) that, given a fixed obstruction, the probability that a random h confirms the obstruction is very small. This is captured by the two lemmas below.

Lemma 13. *The obstruction constructed in Section 3.2 satisfies the conditions on $h = \widehat{h} \circ \widetilde{h}$.*

Proof. Since $W\backslash\{x_w\} \subseteq Z\backslash\{x^*\}$, the choice of Z implies that either $W \subseteq Q$ or $Q \cup W\backslash\{x_w\}$ is d-independent.

The matchings M_i were all submatchings of perfect i-matchings. We also picked M_i is minimally i-dependent, but for an i-matching M_i this also implies that M_i is minimally $(i-1)$-dependent, and therefore $M_i\backslash\{e_i\}$ is $(i-1)$-independent. Finally, we constructed L_i and Z_i to be i-zero. ☐

Lemma 14. *Given the objects of an obstruction, the probability that $h = \widehat{h} \circ \widetilde{h}$ confirms the obstruction is at most*

$$\left(\prod_{x \in W\backslash\{x_w\}} p_x\right) \Big/ |\Sigma|^{w-2+\sum_{i=1}^{d-2}(|M_i|-1)}.$$

Proof. For simplicity, we group the conditions above in the following events: we define the event $\mathcal{C}_S$ to be the event that $W\backslash\{x_w\} \subseteq X^{f,h}$ given that the set $W\backslash\{x_w\}$ is d-independent. Then, for each $i \in \{1, \dots, d-1\}$, we define $\mathcal{C}^{(i)}$ to be the event that $M_i \backslash e_i$ is an i-matching conditioned on the fact that it is $(i-1)$-independent. Finally, we define the last event $\mathcal{C}^{(d)}$ to be the event that M_d is a d-matching conditioned on the fact that it is $(d-1)$-independent. It is then sufficient to show that:

$$\Pr\left(\mathcal{C}_S \wedge \bigwedge_{i=1}^{d} \mathcal{C}^{(i)}\right) \leqslant \left(\prod_{x \in W\backslash\{x_w\}} p_x\right) \Big/ |\Sigma|^{w-2+\sum_{i=1}^{d-2}(|M_i|-1)}.$$

We proceed from the bottom up, in the following sense. For every $i \in \{1, \dots, d-1\}$, the randomness of the event $\mathcal{C}^{(i)}$ conditioned on $\bigwedge_{j=1}^{i-1} \mathcal{C}^{(j)}$ depends solely on $\widetilde{h}_i$. We invoke Lemma 10 for $M_i \backslash e_i$ and get that

$$\Pr\left(\mathcal{C}^{(i)} \;\middle|\; \bigwedge_{j=1}^{i-1} \mathcal{C}^{(j)}\right) \leqslant 1/\left|\Sigma\right|^{|M_i|-1} .$$

When it comes to level d, from Lemma 10 applied to M_d, we get that the probability of $\mathcal{C}^{(d)}$ conditioned on $\bigwedge_{j=1}^{d-1} \mathcal{C}^{(j)}$ is at most $1/|\Sigma|^{|M_d|}$. We now notice that $|M_{d-1}| + |M_d| = w - 1$ by construction, and so:

$$\Pr\left(\bigwedge_{j=1}^{d} \mathcal{C}^{(j)}\right) \leqslant 1/|\Sigma|^{w-2+\sum_{i=1}^{d-2}(|M_i|-1)} \ .$$

Finally, we know that either $W \subseteq Q$ or $Q \cup W\backslash\{x_w\}$ is d-independent. If $W \subseteq Q$, then $p_x = 1$ for all $x \in W$, and therefore, trivially,

$$\Pr\left(\mathcal{C}_S \;\middle|\; \bigwedge_{j=1}^{d} \mathcal{C}^{(j)}\right) \leqslant \prod_{x \in W\backslash\{x_w\}} p_x = 1 \ .$$

Otherwise $Q \cup W\backslash\{x_w\}$ is d-independent. This means that the derived keys $\widetilde{h}(Q \cup W\backslash\{x_w\})$ are linearly independent. We can then apply Lemma 9 to this set of derived keys and the final simple tabulation hash function $\widehat{h}$. We get that the final hash values $h(Q \cup W\backslash\{x_w\})$ are chosen independently and uniformly at random. For any one element x, by definition, $\Pr\left(x \in X^{f,h}\right) \leqslant p_x$ and so:

$$\Pr\left(\mathcal{C}_S \;\middle|\; \bigwedge_{j=1}^{d} \mathcal{C}^{(j)}\right) \leqslant \prod_{x \in W\backslash\{x_w\}} p_x \ .$$

This completes the claim. $\square$

4 Simplified analysis

In this section, we present a simplified analysis showing that, under the hypotheses of our theorem, $\widetilde{h}(X^{f,h})$ is linearly dependent with probability at most $\Theta(\mu^3(17/|\Sigma|)^d) + 2^{-|\Sigma|/8}$. Later, we will replace $(17/|\Sigma|)^d$ with $(3/|\Sigma|)^d$, which essentially matches the growth in our lower bound.

For now we use a fixed limit $w_0 = |\Sigma|/2^{5/2}$ on the set size $w = |W|$. With this limited set size, we will derive the $\Theta(\mu^3(17/|\Sigma|)^d)$ bound. The $2^{-|\Sigma|/8}$ bound will stem for sets of bigger size and will be derived in a quite different way.

Our goal is to study the probability that there exists a combinatorial obstruction agreeing with a random tornado hash function h; if not, $\widetilde{h}(X^{f,h})$ is linearly independent. To do this, we consider a union bound over all combinatorial obstructions as such:

$$\sum_{W,x_w,M_d,M_{d-1},e_{d-1},L_{d-1},Z_{d-1},\ldots M_1,e_1,L_1,Z_1} \Pr_h[h \text{ confirms the obstruction}] \tag{4}$$

The above sum is informally written in that we assume that each element respects the previous elements of the obstruction, e.g., for $i < d-2$, M_i is a greedy matching over Z_{i+1}. Likewise, in the probability term, it is understood that h is supposed to confirm the obstruction whose combinatorial description is $(W, x_w, M_d, M_{d-1}, e_{d-1}, L_{d-1}, Z_{d-1}, \ldots M_1, e_1, L_1, Z_1)$.

Using Lemma 14, we bound (4) by

$$\sum_{W,x_w,M_d,M_{d-1},e_{d-1},L_{d-1},Z_{d-1},\ldots M_1,e_1,L_1,Z_1} \left(\prod_{x \in W \setminus \{x_w\}} p_x \right) \Big/ |\Sigma|^{w-2+\sum_{i=1}^{d-2}(|M_i|-1)}$$

$$\leqslant \left(\sum_{W,x_w,M_d,M_{d-1},e_{d-1},L_{d-1},Z_{d-1}} \left(\prod_{x \in W \setminus \{x_w\}} p_x \right) |\Sigma|^{2-w} 2^{w/4-1} \right) \tag{5}$$

$$\times \prod_{i=1}^{d-2} \max_{Z_{i+1}} \left(\sum_{M_i,e_i,L_i,Z_i} |\Sigma|^{1-|M_i|} \Big/ 2^{(|Z_{i+1}|-|Z_i|)/4} \right). \tag{6}$$

Above , $W, x_w, M_d, M_{d-1}, e_{d-1}, L_{d-1}, Z_{d-1}$ are all the elements from the top two levels and we refer to (5) as the "top" factor. We refer to Equation (6) as the "bottom" factor which is the product of "level" factors.

For each lower level $i \leqslant d-2$, the elements M_i, e_i, L_i, Z_i are only limited by Z_{i+1}, so for a uniform bound, we just consider the maximizing choice of Z_{i+1}. For this to be meaningful, we divided the level factor (6) by $2^{(|Z_{i+1}|-|Z_i|)/4}$ and multiplied (5) by $2^{w/4-1} \geqslant \prod_{i=1}^{d-2} 2^{(|Z_{i+1}|-|Z_i|)/4}$. This inequality follows because $|Z_{d-1}| \leqslant w$ and $|Z_1| \geqslant 4$. The exponential decrease in $|Z_{i+1}|$ helps ensuring that the maximizing choice of Z_{i+1} has bounded size (and is not infinite). We note here that our bound $|W| \leqslant w_0 = |\Sigma|/2^{5/2}$ is only needed when bounding the level factors, where it implies that $|Z_{i+1}| \leqslant w_0$.

4.1 The top two levels and special indexing for matchings and zero sets

We now wish to bound the top factor (5) from the two top levels. Below, we will first consider w fixed. Later we will sum over all relevant values of w.

We want to specify the w keys in W using the fact that the keys from $W \setminus \{x_w\}$ hash independently. Thus, we claim that we only have to specify the set $V = W \setminus \{x_w\}$, getting x_w for free. By construction, we have x_w in the zero-set Z_{d-1}, so

$$x_w = \triangle(Z_{d-1} \setminus \{x_w\}). \tag{7}$$

To benefit from this zero-set equality, we need the special ordering $\preceq$ from the construction. It uses the standard ordering $\leqslant$ of V since all these keys are known, and then it just places the yet unknown key x_w last. The special ordering yields and indexing $x_1 \prec x_2 \prec \cdots \prec x_w$ (this is not the order in which we traversed the cycles in the construction except that x_w is last in W in both cases). Formally, we can now specify all the M_i, L_i, Z_i over the index set $\{1, \ldots, w\}$. For $i < w$, we directly identify x_i as the ith element in V. This way we identify all $x_i \in Z_{d-1} \setminus \{x_w\}$, and then we can finally compute the special last key x_w; meaning that we completely resolve the correspondance between indices and keys in W. As a result, we can bound (5) by

$$\sum_{V:|V|=w-1} \left(\prod_{x \in V} p_x \right) \times \sum_{M_d,M_{d-1},e_{d-1},L_{d-1},Z_{d-1}} |\Sigma|^{2-w} 2^{w/4-1}.$$

For the first part, with $v = w-1$, we have

$$\sum_{V:\,|V|=v} \left(\prod_{x \in V} p_x \right) = \frac{1}{v!} \sum_{(x_1,\ldots,x_v)} \prod_{i=1}^{v} p_{x_i} \leqslant \frac{1}{v!} \prod_{i=1}^{v} (\sum_x p_x) = \frac{\mu^v}{v!}. \tag{8}$$

For the second part, we note M_{d-1} and M_d can both be chosen in $(w-1)!!$ ways and we know that e_{d-1} is the edge in M_{d-1} incident to w. Since the submatching L_{d-1} of M_{d-1} contains the known e_{d-1}, it can be chosen in at most $2^{|M_{d-1}|-1} = 2^{w/2-1}$ ways. Putting this together, we bound (5) by

$$\frac{\mu^{w-1}}{(w-1)!}((w-1)!!)^2\, 2^{w/2-1}|\Sigma|^{-w+2}2^{w/4-1} = \frac{\mu^{w-1}}{|\Sigma|^{w-2}} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot 2^{3w/4-2}$$
$$\leqslant \frac{\mu^3}{|\Sigma|^2} 2^{4-w} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot 2^{3w/4-2} = \frac{\mu^3}{|\Sigma|^2} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot 2^{2-w/4} \tag{9}$$

Above we got to the second line using our assumption that $\mu \leqslant |\Sigma|/2$. This covers our top factor (5), including specifying the set Z_{d-1} that we need for lower levels.

Union bound over all relevant sizes. We will now sum our bound (9) for a fixed set size w over all relevant set sizes $w \leqslant w_0$, that is, $w = 4, 6, \ldots, w_0$. The factor that depends on w is

$$f(w) = \frac{(w-1)!!}{(w-2)!!}/2^{w/4}.$$

We note that $f(w+2) = f(w)\frac{w+1}{\sqrt{2}w}$ and $\frac{w+1}{\sqrt{2}w} < 4/5$ for $w \geqslant 8$, so

$$\sum_{\text{Even } w=4}^{w_0} f(w) < f(4) + f(6) + 5f(8) < 4.15.$$

Thus we bound the top factor (5) over all sets W of size up to w_0 by

$$\sum_{\text{Even } w=4}^{w_0} \left(\frac{\mu^3}{|\Sigma|^2} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot 2^{2-w/4} \right) < 16.6(\mu^3/|\Sigma|^2). \tag{10}$$

4.2 Lower levels with greedy matchings

We now focus on a lower level $i \leqslant d-2$ where we will bound the level factor

$$\max_{Z_{i+1}} \left(\sum_{M_i, e_i, L_i, Z_i} |\Sigma|^{1-|M_i|} \Big/ 2^{(|Z_{i+1}|-|Z_i|)/4} \right). \tag{11}$$

We want a bound of $O(1/|\Sigma|)$, implying a bound of $O(1/|\Sigma|)^{d-2}$ for all the lower levels in (6).

All that matters for our bounds is the cardinalities of the different sets, and we set $m_i = |M_i|$ and $z_i = |Z_i|$. For now we assume that $z_{i+1} \leqslant w_0$ and m_i are given.

Lemma 15. *With a given linear ordering over a set S of size n, the number of greedy matchings of size k over S is $(n-1)^{\underline{\underline{k}}}$.*

Proof. We specify the edges one at the time. For greedy matchings, when we pick the jth edge, the first end-point is the smallest free point in S and then there are $n-2j+1$ choices for its match, so the number of possible greedy matchings of size k is $(n-1)^{\underline{\underline{k}}}$. $\square$

Recall that e_i denotes the last edge in our greedy matching M_i and that $e_i \in L_i$. In our case, we are only going to specify $M_i' = M_i \backslash \{e_i\}$ and $L_i' = L_i \backslash \{e_i\}$. The point is that if we know M_i' and L_i', then we can compute e_i. More precisely, we know that e_i is the next greedy edge to be added to M_i', so we know its first end-point x. We also know that $Z_i = \bigcup L_i$ is a zero-set, so the other end-point can be computed as key

$$y = \triangle(\{x\} \cup \bigcup L_i') \tag{12}$$

Above we note that even though the keys in x and L_i' are only specified as indices, we know how to translate between keys and indices, so we can compute the key y and then translate it back to an index.

By Lemma 15, we have $(z_{i+1}-1)^{\underline{\underline{m_i-1}}}$ choices for M_i', and then there are 2^{m_i-1} possible submatchings $L_i' \subset M_i'$. The number of combinations for M_i, e_i, L_i is thus bounded by

$$(z_{i+1}-1)^{\underline{\underline{m_i-1}}} \cdot 2^{m_i-1}.$$

We want to multiply this by

$$|\Sigma|^{1-m_i}/2^{(z_{i+1}-z_i)/4}.$$

Here $z_i = 2|L_i| \leqslant 2m_i$, so we get a bound of

$$(z_{i+1}-1)^{\underline{\underline{m_i-1}}} \cdot (2^{3/2}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4-1/2} \leqslant 2^{1/2}(2^{3/2}z_{i+1}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4}. \tag{13}$$

We now note that

$$(2^{3/2}z_{i+1}/|\Sigma|) \leqslant 1/2.$$

since $z_{i+1} \leqslant w \leqslant w_0 = |\Sigma|/2^{5/2}$. Having this factor bounded below 1 is critical because it implies that the term decreases as m_i grows.

Still keeping z_{i+1} fixed, we will now sum over all possible values of m_i. Since M_i contains a zero-matching L_i of size at least 2, that is 2 edges covering 4 keys, we have that $m_i \geqslant 2$. Moreover, the terms of the summation are halving, hence they sum to at most twice the initial bound, so we get

$$\sum_{m_i \geqslant 2} 2^{1/2}(2^{3/2}z_{i+1}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4} \leqslant 2^{3/2}(2^{3/2}z_{i+1}/|\Sigma|)/2^{z_{i+1}/4}. \tag{14}$$

The maximizing real value of z_{i+1} is $4/\ln 2 = 5.77$, but z_{i+1} also has to be an even number, that is, either 4 or 6, and 6 yeilds the maximum bound leading to the overall bound of $16.98/|\Sigma|$ for (6). Together with our bound (10), we get a total bound of

$$16.6(\mu^3/|\Sigma|^2) \cdot (17/|\Sigma|)^{d-2}. \tag{15}$$

4.3 Large set sizes

We now consider sets W of sizes $w > w_0$. In this case, we will only consider the two top levels of the obstructions. Recall that we have a cycle traversal $x_1, \dots, x_w$. If $w > w_0$, we only use the prefix $x_1, \dots, x_{w_0}$, where we require that w_0 is even. The two matchings M_d and M_{d-1} are reduced accordingly. Then M_{d-1} is a perfect matching on $W_0 = \{x_1, \dots, x_{w_0}\}$ while M_d is maximal excluding x_{w_0}. Each of these matchings can be chosen in $(w_0-1)!!$ ways.

This time we know that we have full independence. We know that $\widetilde{W}_0$ is a strict subset of the original minimimal zero set $\widetilde{Z}$ of derived keys, so the keys in W_0 all hash independently. Therefore the probability that they are all selected is bounded by $\prod_{x \in W_0} p_x$.

Also, since we terminate the traversal early, we know that M_{d-1} is $(d-2)$-linearly independent, and M_d must be $d-2$-linearly independent, so the probability that these two matchings are satisfied is $1/|\Sigma|^{|M_{d-1}|+|M_d|} \leqslant 1/|\Sigma|^{w_0-1}$.

The total bound for all $w > w_0 = |\Sigma|/2^{5/2}$ is now

$$\sum_{W_0\colon |W_0|=w_0, M_d, M_{d-1}} \left(\prod_{x \in W_0} p_x\right) /|\Sigma|^{w_0-1} \leqslant \frac{\mu^{w_0}}{w_0!}((w_0-1)!!)^2/|\Sigma|^{w_0-1} \leqslant w_0|\Sigma|/2^{w_0} \tag{16}$$

$$\leqslant 1/2^{|\Sigma|/8}. \tag{17}$$

The last step holds easily for $|\Sigma| \geqslant 256$. We note that (16) holds for any choice of w_0, limiting the probability of any obstruction with $|W| < w$.

5 Tighter analysis

We will now tighten the analysis to prove that

Theorem 5. *Let $h : \Sigma^c \to \mathcal{R}$ be a tornado tabulation hash function with d derived characters and f be an s-selector as described above. If $\mu^f \leqslant \Sigma/2$, then $h^{(t)}$ is fully random on $X^{f,h^{(s)}}$ with probability at least*

$$1 - \mathsf{DependenceProb}(\mu^f, d, \Sigma) .$$

5.1 Bottom analysis revisited

We first tigthen the analysis of the bottom factor (6) so as to get a bound of $O((3/\Sigma)^{d-2})$ and, together with our top bound (10), obtain an overall bound of $O(\mu^3(3/\Sigma)^d)$ matching our lower bound within a constant factor. We are still using our assumption that $w \leqslant w_0 \leqslant |\Sigma|/2^{5/2}$ implying that $z_{i+1} \leqslant |\Sigma|/2^{5/2}$.

First we look at a single level i. For a tighter analysis of the level factor (11), we partition into cases depending on z_{i+1} and to some degree on $z_i = 2m_i$. Recall that z_{i+1} is given from the level above.

If $z_{i+1} \leqslant 6$, then we must have $z_i = z_{i+1}$, since we cannot split into two zero sets each of size at least 4. This implies that the factor $2^{|Z_{i+1}|-|Z_i|}$ is just one. Also, in this case, $M_i = L_i$ must be a perfect matching on $Z_{i+1} = Z_i$, and such a perfect matching can be in $(z_{i+1}-1)!!$ ways. Thus, for given $z_{i+1} \leqslant 6$, we bound the level factor by

$$(z_{i+1}-1)!!/|\Sigma|^{z_{i+1}/2-1} \leqslant 3/|\Sigma|. \tag{18}$$

with equality for $z_{i+1} = 4$. As a result, if $z_{d-1} \leqslant 6$, then we have already achieved a bound of $(3/|\Sigma|)^{d-2}$ for the bottom factor.

Split levels. We now consider $z_{i+1} \geqslant 8$. Now it is possible that Z_{i+1} can split into two sets so that $z_i < z_{i+1}$. For a given m_i and z_{i+1}, we already had the bound (13)

$$2^{1/2}(2^{3/2}z_{i+1}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4}.$$

Since $(2^{3/2}z_{i+1}/|\Sigma|) \leqslant 1/2$, the worst case is achieved when $m_i = 2$, but here we can do a bit better. In (13) we had a factor 2^{m_i-1} to specify the subset L_i of M_i containing e_i, but since L_i should have size at least 4, we have $L_i = M_i$. Thus, for $m_i = 2$ and given z_{i+1}, we improve (13) to

$$2(z_{i+1}/|\Sigma|)/2^{z_{i+1}/4}.$$

For $z_{i+1} \geqslant 8$, this is maximized with $z_{i+1} = 8$. Thus for $m_i = 2$ and $z_{i+1} \geqslant 8$, the level factor (11) is bounded by

$$(2 \cdot 8/|\Sigma|)/2^2 = 4/|\Sigma|.$$

Now, for $m_i \geqslant 3$, we just use the bound (13). As in (14), we get

$$\sum_{m_i \geqslant 3} 2^{1/2}(2^{3/2}z_{i+1}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4} \leqslant 2 \cdot 2^{1/2}(2^{3/2}z_{i+1}/|\Sigma|)^2/2^{z_{i+1}/4}$$

Over the reals, this is maximized with $z_{i+1} = 2 \cdot 4/\ln 2 \approx 11.52$, but we want the maximizing even z_{i+1} which is 12, and then we get a bound of $1.6/|\Sigma|$.

We now want to bound the whole bottom factor in the case where $z_{d-1} \geqslant 8$. Let j be the lowest level with $z_{j+1} \geqslant 8$. For all lower levels, if any, the level factor is bounded by $(3/|\Sigma|)$. Also, for all higher levels $i > j$, we have $2m_i = z_i \geqslant 8$, hence $m_i \geqslant 4$, so the level factor for higher levels is bounded by our last $1.6/|\Sigma|$. The worst case is the level j, where we could get any m_s (note that if $s = 1$, we have no guarantee that $m_i \leqslant 2$), hence we have to add the bound $4/|\Sigma|$ for $m_s = 2$ with the bound $1.6|\Sigma|$ for $m_s \geqslant 3$, for a total bound of $5.6/|\Sigma|$.

Thus, for a given j, the bottom factor is bounded by

$$(3/|\Sigma|)^{j-1}(5.6/|\Sigma)(1.6/|\Sigma|)^{d-2-j} = (3/|\Sigma|)^{d-2} * (5.6/3) * (1.6/3)^{d-2-j}.$$

Summing, this over $j = 1, \ldots, d-2 \geqslant 1$, we get a bound of at most

$$(3/|\Sigma|)^{d-2}(5.6/3)(1/(1-1.6/3)-1) < 4(3/|\Sigma|)^{d-2}\ . \tag{19}$$

This is our bound for the whole bottom factor when we maximized with $z_{d-1} \geqslant 8$. Since it is larger than our bound of $3/|\Sigma|)^{d-2}$ when we maximized with $z_{d-1} \leqslant 6$, we conclude that it is also our bound if we maximize over any value of z_{d-1}. In combination with our top factor, $16.6(\mu^3/|\Sigma|^2)$ from (10), we get a combined bound of

$$16.6(\mu/|\Sigma|^2)4(3/|\Sigma|)^{d-2} \leqslant 7\mu^3(3/|\Sigma|)^d. \tag{20}$$

5.2 Increasing the maximization range

We will now show how to deal with larger sets up to size $w_0^+ = 0.63|\Sigma|$. The level factor with a fixed z_{i+1} and m_i has the following tight version from (13)

$$\begin{aligned}&(z_{i+1}-1)^{\underline{\underline{m_i-1}}} \cdot (2^{3/2}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4-1/2}\\ &= 2^{1/4}f(m_i-1, z_{i+1}-1) \text{ where } f(x,y) = y^{\underline{\underline{x}}} \cdot (2^{3/2}/|\Sigma|)^x/2^{y/4}.\end{aligned}$$

We want to find the sum over relevant m_i with the maximizing z_i. However, we already considered all even $z_{i+1} \leqslant |\Sigma|/2^{5/2}$, so it suffices to consider $z_{i+1} \in (|\Sigma|/2^{5/2},\, |\Sigma|/2]$. Also, for a given z_{i+1}, we have to sum over all $m_i \in [2, z_{i+1}/2]$. Thus we want to bound

$$\max_{\text{odd } y \in [|\Sigma|/2^{5/2},\, |\Sigma|/2)} \sum_{x=1}^{\lfloor y/2 \rfloor} f(x, y).$$

Note that $f(x+1, y) = f(x, y) \cdot (y-2x)(2^{3/2}/|\Sigma|)$. Hence $f(x+1, y) < f(x, y) \iff y-2x < |\Sigma|/2^{3/2}$.

Assume that $y \geqslant |\Sigma|/2^{3/2}$ and consider the smallest integer x_y such that $y - 2x_y < |\Sigma|/2^{3/2}$. For a given value of y this x_y maximizes $f(x_y, y)$.

To bound $f(x_y, y)$, we note that

$$y^{\underline{x}} < (y - x + 1)^x$$

We have $y < w_0^+ = 0.63|\Sigma|$ and $y - 2x_y < |\Sigma|/2^{3/2}$, so

$$y - x_y + 1 \leqslant (0.63 + 1/2^{3/2})|\Sigma|/2 + 1 \leqslant |\Sigma|/2.$$

Therefore

$$f(x_y, y) \leqslant y^{\underline{x_y}} \cdot (2^{3/2}/|\Sigma|)^{x_y}/2^{y/4} \leqslant ((y - x_y + 1)2^{3/2}/|\Sigma|)^{x_y}/2^{y/4} \leqslant 1/2^{(y-2x_y)/4}.$$

We also have $y - 2(x_y - 1) \geqslant |\Sigma|/2^{3/2}$, so we get

$$f(x_y, y) \leqslant 1/2^{(|\Sigma|/2^{3/2}-2)/4} = 1/2^{|\Sigma|/2^{7/2}-1/2}.$$

For $|\Sigma| \geqslant 256$, this is below $0.015/|\Sigma|^2$, so even if we sum over all $x \leqslant y/2 \leqslant |\Sigma|/2$, we get a bound below $0.008/|\Sigma|$, that is,

$$\max_{y \in [|\Sigma|/2^{3/2},\, w_0^+]} \sum_{x=1}^{\lfloor y/2 \rfloor} f(x, y) \leqslant 0.08/|\Sigma|.$$

Now consider $y < |\Sigma|/2^{3/2}$. Then $f(x, y)$ is decreasing in x starting $f(0, y) = 1/2^{y/4}$. Summing over all $x \leqslant y/2$, we get $(y/2)/2^{y/4}$. This expression is maximized with $y = 4/\ln 2$, so for $y \geqslant |\Sigma|/4$, we get a maximum of $(|\Sigma|/8)/2^{|\Sigma|/16}$. Now $2^{|\Sigma|/16} \geqslant |\Sigma|^2$ for $|\Sigma| \geqslant 256$, so we end up with

$$\max_{y \in [|\Sigma|/4|, |\Sigma|/2^{3/2}]} \sum_{x=1}^{\lfloor y/2 \rfloor} f(x, y) \leqslant 1/(8|\Sigma|).$$

Finally we consider $y \in [|\Sigma|/2^{5/2},\, |\Sigma|/4]$. Then $f(x + 1, y) \leqslant f(x, y)(y\, 2^{3/2}/|\Sigma|) \leqslant f(x, y)/2^{1/2}$, so

$$\sum_{x=0}^{\infty} f(x, y) \leqslant f(0, y)/(1 - 2^{-1/2}) \leqslant 1/((1 - 2^{-1/2})2^{y/4}).$$

With $y \geqslant |\Sigma|/2^{5/2}$, this is maximized with $y = |\Sigma|/2^{5/2}$, so we get the bound

$$\max_{y \in [|\Sigma|/2^{5/2},\, |\Sigma|/4]} \sum_{x=0}^{\infty} f(x, y) \leqslant 1/((1 - 2^{-1/2})2^{(|\Sigma|/2^{9/2})}) \leqslant 0.344/|\Sigma|.$$

Putting all our bounds together, we conclude that

$$\max_{\text{odd } y\in[|\Sigma|/2^{5/2},\, w_0^+)} \sum_{x=1}^{\lfloor y/2\rfloor} f(x,y) \leqslant 0.344/|\Sigma|.$$

Our bound for the level factor with $z_{i+1} < (|\Sigma|/2^{5/2}, |\Sigma|/2]$ is $2^{1/2}$ times bigger, but this is still much smaller than all the bounds from Section 5.1 based on smaller z_{i+1}. Thus we conclude that the analysis from Section 5.1 is valid even if allow z_{i+1} to go the whole way up to $w_0^+ = 0.63|\Sigma|$. Therefore (20) bounds the probability of any obstruction with set size $|W| \leqslant w_0^+$.

However, for the probability of obstructions with sets sizes $|W| > w_0^+$, we can apply (16), concluding that they happen with probability bounded by

$$w_0^+|\Sigma|/2^{w_0^+} = 0.63|\Sigma|^2/2^{0.63|\Sigma|} < 1/2^{|\Sigma|/2}.$$

The last step used $|\Sigma| \geqslant 256$. Together with (20) we have thus proved that the probability of any obstruction is bounded by

$$7\mu^3(3/|\Sigma|)^d + 1/2^{|\Sigma|/2}. \tag{21}$$

This is for simple tornado hashing without the twist.

5.3 Tornado hashing including the twist

We will now return to the original tornado hashing with the twisted character

$$\tilde{x}[c] = x[c] \oplus \widetilde{h}_0(x[< c]).$$

This twist does not increase the number of lookups: it is still $c + d$ with c input characters and d derived characters, so the speed should be almost the same, but we will gain a factor $3/|\Sigma|$ in the probability, like getting an extra derived character for free.

The obstruction is constructed exactly as before except that we continue down to level 0 rather than stopping at level 1. All definitions for level i are now just applied for $i = 0$ as well. However, we need to reconsider some parts of the analysis. First, we need to prove that Lemma 10 also holds for $i = 0$:

Lemma 10 *Let M be a partial matching on Σ^c. Conditioning on M being $(i-1)$-independent, M is an i-matching with probability $1/|\Sigma|^{|M|}$.*

Proof. Since M is (-1)-independent, we know that

$$\mathrm{DiffKeys}(M, -1) = \{(\tilde{x} \triangle \tilde{y})[< c] \mid \{x, y\} \in M\}$$

is linearly independent. We note here that $(\tilde{x} \triangle \tilde{y})[< c] = (x \triangle y)[< c]$. We want to know the probability that M is a 0-matching, that is, the probability that $\tilde{x}[c] = \tilde{y}[c]$ for each $\{x, y\} \in M$. For $i > 0$, we had

$$\tilde{x}[c + i] = \tilde{y}[c + i] \iff \widetilde{h}_i((\tilde{x} \triangle \tilde{y})[< c + i]) = 0.$$

However, for $i = 0$, we have

$$\tilde{x}[c] = \tilde{y}[c] \iff \tilde{h}_0((\tilde{x} \triangle \tilde{y})[< c]) = x[c] \oplus y[c].$$

However, Lemma 9 states that the simple tabulation hash function $\widetilde{h}_0$ is fully random on the linearly indepedent DiffKeys$(M, -1)$, so we still conclude that M is a 0-matching with probability $1/\Sigma^{|M|}$. □

There is one more thing to consider. We have generally used that if Z is an i-zero set, that is, if $\widetilde{Z}[\leqslant c+i]$ is a zero-set, then Z is also a zero-set. However, this may no longer be the case. All we know is that $\widetilde{Z}[\leqslant c]$ is a zero-set. This also implies that $Z[< c] = \widetilde{Z}[< c]$ is a zero-set. However, we claim that

$$\bigoplus Z = 0 \tag{22}$$

Here $\bigoplus Z$ is xoring that keys in Z viewed as bit-strings. Note that $\triangle Z = \varnothing$ implies $\bigoplus Z = 0$. Since $Z[< c]$ is a zero set, we know that $\{\widetilde{h}_0(x[< c]) \mid x \in Z\}$ is a zero set. We also know that $\widetilde{Z}[c]$ is a zero set and it is equal to $\{\widetilde{h}_0(x[< c]) \oplus x[c] \mid x \in Z\}$. Thus we have

$$\bigoplus\{\widetilde{h}_0(x[< c]) \oplus x[c] \mid x \in Z\} = 0 = \bigoplus\{\widetilde{h}_0(x[< c]) \mid x \in Z\}$$

implying $\bigoplus\{x[c] \mid x \in Z\} = 0$. Together with $Z[< c]$ being a zero set, this settles (22). As a result, for the coding key coding in (7) and (12), we just have to replace $\triangle$ with $\bigoplus$.

No other changes are needed. Level 0 gives exactly the same level factor (11) as the levels $i > 0$, so it is like getting an extra level for free. Therefore with tornado hashing we improve the bottom factor for simple tornado hashing (19) to $4(3/|\Sigma|)^{d-1}$ and the probability of any small obstruction from (20) to $7\mu^3(3/|\Sigma|)^{d+1}$. The probability of any obstruction is thus improved from (21) to

$$7\mu^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}. \tag{23}$$

5.4 Full randomness for large sets of selected keys

While implementing tabulation hashing we often want to set the character size so that all tables fit in cache. Indeed, this is the main reason why tabulation-based hashing schemes are extremely fast in practice. However, restricting the size of Σ constrains how many keys we can expect to be hashed fully randomly: in particular, Theorem 5 states that a set of characters Σ allows for up to $|\Sigma|/2$ selected keys to hash fully randomly with high probability. Most experiments on tabulation-based hashing [1, 2], though, use 8-bit characters (namely $|\Sigma| = 2^8$). This implies that whenever the selected keys are $s \leqslant 2^7$ then they hash uniformly at random with high probability. However, we may want local full randomness for s keys, where $s \gg 2^7$. A trade-off between memory usage and number of keys hashed uniformly is of course unavoidable, however we can improve over the one in Theorem 5 with a clever observation.

More precisely, Theorem 5 states that given a set $X^{f,h}$ of query-selected keys with $\mu^f \leqslant |\Sigma|/2$ their derived keys are linearly dependent with probability at most $\mathsf{DependenceProb}(\mu, d, \Sigma) = 7\mu^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}$ while tornado is using $O(c|\Sigma|)$ words of memory and exactly $c + d$ lookups in tables of size $|\Sigma|$. Recall that μ^f here is the expected size of $X^{f,h}$ for a fully random h, when the queries are chosen by an adaptive adversary. If we want to handle larger sets of selected keys with $\mu^f \gg |\Sigma|/2$ using Theorem 5, then we need to use $O(c + d)$ larger tables of size roughly $2\mu^f$. The clever observation we make here is that, in order to obtain a meaningful probability bound, it is not necessary at all to employ $O(c + d)$ of such larger tables. Indeed, we just need the tables in the top two levels to have size $|\Psi| \geqslant 2\mu^f$ to obtain that the derived keys are linearly dependent with probability at most

$$14|X|^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} + 1/2^{|\Sigma|/2}$$

losing a factor two with respect to Theorem 5. We name this variant of tornado tabulation tornado-mix tabulation, because the last two derived characters can be evaluated in parallel as in mixed tabulation hashing [10].

Formal definition of tornado-mix. For each $i = 0, \ldots, d-2$ we let $\widetilde{h}_i : \Sigma^{c+i-1} \longrightarrow \Sigma$ be a simple tabulation hash function. For $i = d-1, d$ we let $\widetilde{h}_i : \Sigma^{c+d-2} \longrightarrow \Psi$ be a simple tabulation hash function. Given a key $x \in \Sigma^c$, we define its *derived key* $\widetilde{x} \in \Sigma^{c+d-2} \times \Psi^2$ as $\widetilde{x} = \widetilde{x}_1 \cdots \widetilde{x}_{c+d}$, where

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i = 1, \ldots, c-1 \\ x_c \oplus \widetilde{h}_0(\widetilde{x}_1 \ldots \widetilde{x}_{c-1}) & \text{if } i = c \\ \widetilde{h}_{i-c}(\widetilde{x}_1 \ldots \widetilde{x}_{i-1}) & \text{if } i = c+1, \ldots, c+d-2. \\ \widetilde{h}_{i-c}(\widetilde{x}_1 \ldots \widetilde{x}_{c+d-2}) & \text{if } i = c+d-1, c+d. \end{cases}$$

Finally, we have a simple tabulation hash function $\widehat{h} : \Sigma^{c+d-2} \times \Psi^2 \longrightarrow \mathcal{R}$, that we apply to the derived key. The *tornado-mix tabulation* hash function $h : \Sigma^c \longrightarrow \mathcal{R}$ is then defined as $h(x) = \widehat{h}(\widetilde{x})$.

Cache efficiency. It is worth to notice that such construction allows us to store in cache all $|\Sigma|$-sized tables while the two $|\Psi|$-sized tables might overflow cache. However, these larger tables are accessed only once while evaluating tornado and they can be accessed in parallel.

Local uniformity theorem. Now we are ready to state an analogous of Theorem 4 for larger sets of selected keys. In what follows we use f, μ^f and $X^{f,h}$ as defined in Section 1.2.

Theorem 16. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado-mix tabulation hash function with d derived characters, the last two from Ψ, and select function f. If $\mu = \mu^f \leqslant |\Psi|/2$ then the derived selected keys $\widetilde{h}(X^{f,h})$ are linearly dependent with probability at most*

$$14\mu^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} + 1/2^{|\Sigma|/2} .$$

Proof. This proof works exactly as the proof of Theorem 4, except for a few slight differences. We limit ourselves to listing such small differences. In Section 3.2 we define Z as the smallest d-zero set among those d-zero sets minimizing the number of elements not in Q. This definition implies that there exists $x^* \in Z$ such that $Z \backslash \{x^*\}$ is d-independent. Moreover, either $Z \subseteq Q$ or we can choose $x^* \in Z \backslash Q$. In the original proof, we considered the alternating-cycle structure induced by the top level matchings M^*_{d-1} and M^*_d and traversed these cycles leaving x^* last. This ensured that the edges from M^*_d were always "surprising" in the sense that the probability of any such edge being realised by our random choice of h was $1/|\Sigma|$, even after conditioning on all previously discovered edges (these events were, indeed, independent). This observation allowed us to bound the probability of our obstruction being realised by h. In fact, we wanted our obstruction to be constituted by edges which realisations were independent, exluding the last edge. This is exactly what we did in Section 3.2, where all edges but the last edge $e_{d-1} \in M^*_{d-1}$ were realised by h independently.

In the current scenario, it is not obvious that the realisations of traversed edges from M^*_d are all independent and the first edge introducing a dependence belongs to M^*_{d-1}. Here, instead, we traverse the alternating-cycle graph until we find a prefix $W = \{x_1 \dots x_w\}$ such that either (i) M^*_{d-1} restricted to W or (ii) M^*_d restricted to W is $(d-2)$-dependent. Case (i) is identical to the case already analysed, but in case (ii) we need some small changes, essentially swapping the roles of M_d and M_{d-1}.

To understand the interplay between the two cases, note that every time we add a vertex x_w, we add an edge to $M^*_{d-1}|_W$ if w is odd, and to $M^*_d|_W$ if w is even. Case (i) can only happen if w is even while Case (ii) can only happen if w is odd. If we get to $x_w = x^*$, then we have have $W = Z$ and then we are in case (i) since Z is even. Thus, if we end in case (ii), then $x^* \notin W$.

We now consider the slightly reduced traversal sequence where we simply drop the first vertex in the last cycle considered in the traversal. The point is that this vertex was not matched in $M^*_d|W$, but if we skip it then $M^*_d|W$ is a perfect $d-2$-dependent matching while $M^*_{d-1}|W$ is a maximal matching. Since we did not have x^* in W, we have $W \subseteq Z \backslash \{x^*\}$ implying full independence of the hash values over W. Thus, using this reduced traversal sequence, we have swapped the roles of the two top levels. Since we now have two cases, our union based probability bounds are doubled (increasing the leading constant from 7 to 14).

The rest of the analysis is unchanged except that two top levels use the different alphabet Ψ. This means Ψ replaces Σ in our bound (9) for the two top levels. This implies that our overall bound is multiplied by $|\Sigma|^2 / |\Psi|^2$. Combined with the doubling, we get an overall probability bound of

$$14|X|^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} + 1/2^{|\Sigma|/2} .$$

$\square$

From Theorem 16, using Lemma 2, we derive the following analogous of Theorem 5 which was already stated in the introduction. Here we use the same notation as in Theorem 5, where $h^{(s)}$ are the selection bits and $h^{(t)}$ are the free bits.

Theorem 8. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado-mix tabulation hash function with d derived characters, the last two from Ψ, and an s-selector function f. If $\mu = \mu^f \leqslant |\Psi|/2$ then $h^{(t)}$ is fully random on $X^{f,h}$ with probability at least*

$$1 - 14\mu^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} - 1/2^{|\Sigma|/2} .$$

6 Upper Tail Chernoff

In this section, we show a Chernoff-style bound on the number of the selected keys $X^{f,h}$.

Lemma 6. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters, query keys Q and selector function f. Let $\mathcal{I}_{X^{f,h}}$ denote the event that the set of derived selected key $\widetilde{h}(X^{f,h})$ is linearly independent. Then, for any $\delta > 0$, the set $X^{f,h}$ of selected keys satisfies the following:*

$$\Pr\left[\left|X^{f,h}\right| \geqslant (1+\delta)\cdot\mu^f \wedge \mathcal{I}_{X^{f,h}}\right] \leqslant \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{\mu^f} .$$

Proof. We follow the proof of the Chernoff bound for the upper tail. Mainly, we let $\mathcal{J}_x$ denote the indicator random variable for whether the key x gets selected in $X^{f,h}$. Then $\left|X^{f,h}\right| = \sum_{x\in\Sigma^c} \mathcal{J}_x$.

For simplicity, we let $a = 1 + \delta$. Then for any $s > 0$:

$$\begin{aligned}\Pr\Big[\left|X^{f,h}\right| \geqslant a \cdot \mu^f \wedge \mathcal{I}_{X^{f,h}}\Big] &= \Pr\Big[\left|X^{f,h}\right| \cdot [\mathcal{I}_{X^{f,h}}] \geqslant a \cdot \mu^f\Big] \\ &= \Pr\Big[e^{s\left|X^{f,h}\right| \cdot [\mathcal{I}_{X^{f,h}}]} \geqslant e^{sa \cdot \mu^f}\Big] \\ &\leqslant \frac{\mathrm{E}\Big[M_{\left|X^{f,h}\right| \cdot [\mathcal{I}_{X^{f,h}}]}(s)\Big]}{e^{sa \cdot \mu^f}} ,\end{aligned}$$

where $M_{\left|X^{f,h}\right| \cdot [\mathcal{I}_{X^{f,h}}]}(s)$ is the moment generating function of the random variable $(\left|X^{f,h}\right| \cdot [\mathcal{I}_{X^{f,h}}])$, and is equal to:

$$M_{\left|X^{f,h}\right| \cdot [\mathcal{I}_{X^{f,h}}]}(s) = \mathrm{E}\Big[e^{s \cdot \left|X^{f,h}\right| \cdot [\mathcal{I}_{X^{f,h}}]}\Big] = \sum_{i=0}^{\infty} \frac{s^i}{i!} \cdot \mathrm{E}\Big[\left|X^{f,h}\right|^i \cdot [\mathcal{I}_{X^{f,h}}]\Big] .$$

Since $\left|X^{f,h}\right| = \sum_{x \in \Sigma^c} \mathcal{J}_x$, each moment $\mathrm{E}\Big[\left|X^{f,h}\right|^i \cdot [\mathcal{I}_{X^{f,h}}]\Big]$ can be written as the sum of expectations of the form $\mathrm{E}[\prod_{x \in S} \mathcal{J}_x \cdot [\mathcal{I}_{X^{f,h}}]]$, where $S \subseteq \Sigma^c$ is a fixed subset of size $\leqslant i$. In general, the moment generating function can be written as a sum, with positive coefficients, of terms of the form $\mathrm{E}[\prod_{x \in S} \mathcal{J}_x \cdot [\mathcal{I}_{X^{f,h}}]]$ for some fixed subset S. Moreover, each such term amounts to

$$\mathrm{E}\Big[\prod_{x \in S} \mathcal{J}_x \cdot [\mathcal{I}_{X^{f,h}}]\Big] = \Pr\Big[S \subseteq X^{f,h} \wedge \mathcal{I}_{X^{f,h}}\Big] .$$

We now condition on the hash values of the query keys $h|_Q$. For any set $S \subseteq \Sigma^c$, we let $\mathcal{I}_{S \cup Q}$ denote the event that the derived keys in $\widetilde{h}(S \cup Q)$ are linearly independent. Note that $\mathcal{I}_{X^{f,h}}$ being true implies that $\mathcal{I}_{S \cup Q}$ is true. Moreover, since $S \cup Q$ is a fixed (deterministic) set, the event $\mathcal{I}_{S \cup Q}$ only depends on the randomness of $\widetilde{h}$. We get the following:

$$\begin{aligned}\Pr\Big[S \subseteq X^{f,h} \wedge \mathcal{I}_{X^{f,h}} \;\Big|\; h|_Q\Big] &\leqslant \Pr\Big[S \subseteq X^{f,h} \wedge \mathcal{I}_{S \cup Q} \;\Big|\; h|_Q\Big] \\ &= E\Big[\, [\mathcal{I}_{S \cup Q}] \cdot \Pr\Big(S \subseteq X^{f,h} \;\Big|\; \widetilde{h}, h|_Q\Big) \;\Big|\; h|_Q\Big] \\ &= E\Big[\prod_{x \in S} \mathcal{J}^*{}_x \;\Big|\; h|_Q\Big] ,\end{aligned}$$

where $\{\mathcal{J}^*{}_x\}_{x \in X}$ denotes the indicator random variables for choosing to select the keys in S independently and uniformly at random when we fix the hash values of the query keys. This last step is due to the fact that the event if $\mathcal{I}_{S \cup Q}$ is true then, then $(h|_S, h|_Q)$ is fully random and it has the same distribution as $(h^*|_S, h|_Q)$, where h^* is a fully-random hash function. If $\mathcal{I}_{S \cup Q}$ is false, then the entire expression is 0. We can thus continue the proof of Chernoff's as if $\left|X^{f,h}\right|$ were a sum of independent random variables. The claim follows by noticing that, when $\mathcal{I}_{X^{f,h}}$ is true, we have that $E\left[X^{f,h}\right] \leqslant \mu^f$.

$\square$

6.1 Upper Tail Chernoff for larger μ^f

We now consider the case in which we have a selector function for which $\mu^f > |\Sigma|/2$. In this case, even though we cannot guarantee that the set of derived selected keys is linearly independent whp, we show that, whp, its size still cannot be much larger than μ^f. This particular case will be useful in the analysis of linear probing from Section 7.

Lemma 17. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters, query key Q with $|Q| < |\Sigma|/2$, and selector function f such that $\mu^f > |\Sigma|/2$. Then, for any $\delta > 0$, the set of selected derived keys $X^{f,h}$ satisfies the following:*

$$\Pr\left[\left|X^{f,h}\right| \geqslant (1+\delta)\cdot\mu^f\right] \leqslant 4\cdot\left(\frac{e^{\delta_0}}{(1+\delta_0)^{1+\delta_0}}\right)^{|\Sigma|/2} + 4\cdot \textsf{DependenceProb}(|\Sigma|/2, d, \Sigma)\ ,$$

where

$$\delta_0 = \frac{\mu^f}{\mu^f - |Q|}\cdot\frac{|\Sigma|/2 - |Q|}{|\Sigma|/2}\cdot\delta \geqslant \left(1 - \frac{|Q|}{|\Sigma|/2}\right)\cdot\delta\ .$$

Proof. We modify the given selector function f to get another selector function f_p with the same set of query keys Q but with a much smaller μ^{f_p}. Selection according to f_p is done such that, once f selects a key in $\Sigma^c \backslash Q$, f_p further sub-selects it with some probability p. This sub-selection is done independently for every selected key. It follows that, for all $x \in \Sigma^c\backslash Q$, $p_x^{f_p} = p_x^f \cdot p$. Taking into account query keys, we also get that $\mu^{f_p} = (\mu^f - |Q|)\cdot p + |Q| = p\mu^f + (1-p)|Q|$.

Moreover, one can show that, as long as $\mu^{f_p} \leqslant |\Sigma|/2$, all our results about linear independence also hold for such sub-sampled select function. In particular, the only aspect of the proof of Theorem 4 that depends on the probabilities $p_x^{f_p}$ is the proof of Lemma 14. There, we invoke Lemma 9 to get an upper bound on the probability that the set $W\backslash\{x_w\}$ is selected, given that it is d-independent. Notice that, if, additionally, we sub-sample elements from $W\backslash\{x_w\}$ each independently with probability p, we obtain the same bounds as if we initially selected elements with probability $p_x^{f_p}$. Therefore, when $\mu^f > |\Sigma|/2$, we can pick any $p \leqslant (|\Sigma|/2 - |Q|)/(\mu^f - |Q|)$ to get that the set of derived keys for the sampled selection $\widetilde{X}^{f_p,h}$ is indeed linearly independent with probability at least $1 - \textsf{DependenceProb}(\mu^{f_p}, d, \Sigma)$. We use $I_{X^{f_p,h}}$ to denote this event.

The next step is to notice that, conditioned on $\left|X^{f,h}\right| - |Q|$, the distribution of $\left|X^{f_p,h}\right| - |Q|$ is exactly the binomial distribution $B(\left|X^{f,h}\right| - |Q|, p)$. Then, for $p > 1/(\left|X^{f,h}\right| - |Q|)$, we have from [20] that

$$\Pr\Big[\left|X^{f_p,h}\right| \geqslant \mathrm{E}\Big[\left|X^{f_p,h}\right| \mid \left|X^{f,h}\right|\Big] \mid \left|X^{f,h}\right|\Big] > 1/4\ .$$

Therefore, for any $t > 0$:

$$\Pr\Big[\left|X^{f_p,h}\right| \geqslant p\cdot t + (1-p)|Q| \;\Big|\; \left|X^{f,h}\right| \geqslant t\Big] > 1/4\ .$$

We now use this to derive an upper bound on $\Pr\left(\left|X^{f,h}\right| \geqslant t\right)$ as such:

$$\begin{aligned}\Pr\left(\left|X^{f,h}\right| \geqslant t\right) &< 4 \cdot \Pr\left(\left|X^{f_p,h}\right| \geqslant p \cdot t + (1-p)\,|Q| \;\middle|\; \left|X^{f,h}\right| \geqslant t\right) \cdot \Pr\left(\left|X^{f,h}\right| \geqslant t\right) \\ &\leqslant 4 \cdot \Pr\left(\left|X^{f_p,h}\right| \geqslant p \cdot t + (1-p)\,|Q|\right) \\ &\leqslant 4 \cdot \Pr\left(\left|X^{f_p,h}\right| \geqslant p \cdot t + (1-p)\,|Q| \wedge I_{X^{f_p,h}}\right) + \\ &+ 4 \cdot \mathsf{DependenceProb}(\mu^{f_p}, d, |\Sigma|) \;.\end{aligned}$$

We now plug in $t = (1+\delta) \cdot \mu^f$, and get that $p \cdot t + (1-p)\,|Q| \geqslant (1+\delta_0) \cdot \mu^{f_p}$ for

$$\delta_0 \leqslant \frac{p\mu^f}{\mu^{f_p}} \cdot \delta \;.$$

We then invoke Lemma 6 to get that

$$\Pr\left(\left|X^{f_p,h}\right| \geqslant (1+\delta_0) \cdot \mu^{f_p} \wedge I_{X^{f_p,h}}\right) \leqslant \left(\frac{e_0^{\delta}}{(1+\delta_0)^{1+\delta_0}}\right)^{\mu^{f_p}} \;.$$

Finally, we instantiate $p = (|\Sigma|/2 - |Q|)/(\mu^f - |Q|)$ such that $\mu^{f_p} = |\Sigma|/2$. and notice that indeed, $p > 1/(\left|X^{f,h}\right| - |Q|)$ when $\left|X^{f,h}\right| \geqslant (1+\delta) \cdot \mu^f$ and $|Q| < |\Sigma|/2$. We notice that, in this case,

$$\delta_0 = \frac{\mu^f}{\mu^f - |Q|} \cdot \frac{|\Sigma|/2 - |Q|}{|\Sigma|/2} \cdot \delta \geqslant \left(1 - \frac{|Q|}{|\Sigma|/2}\right) \cdot \delta \;.$$

The argument follows.

□

7 Linear Probing with Tornado Tabulation

In this section we show how to formally apply our framework to obtain results on linear probing with tornado tabulation. We present the following main result comparing the performance of linear probing with tornado tabulation to that of linear probing using fully random hashing on a slightly larger keyset.

Theorem 18. *Let $S, S^* \subseteq \Sigma^c$ be sets of keys of size n and $n^* = (1+15\sqrt{\log(1/\delta)/|\Sigma|})n$, respectively, for some $\delta \in (0, 1/6)$. Let T, T^* be arrays of size m, a power of two. Now consider inserting the keys in S (S^*) into T (T^*) with linear probing using tornado tabulation (fully random hashing). Let X and X^* be the number of comparisons performed when inserting a new key x in each of T and T^* (i.e., $x \notin S \cup S^*$). Given the restrictions listed below there exists an event $\mathcal{E}$ with $\Pr(\mathcal{E}) \geqslant 1 - (1/|\Sigma| + 6\delta + 61\log n \cdot \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma))$ such that, conditioned on $\mathcal{E}$, X is stochastically dominated by X^*.*
Restrictions:

- $n/m \leqslant 4/5$
- $|\Sigma| \geqslant 2^{16}$

- $|\Sigma| \geqslant 30 \cdot \log n$
- $\sqrt{\log(1/\delta)/|\Sigma|} \leqslant 1/18$

From Theorem 18, it follows that linear probing using tornado tabulation achieves the same expected number of comparison as in the fully random setting, a proof is given in Section 7.4.

Corollary 19. *Setting $\delta = \Theta(1/|\Sigma|)$, $d \geqslant 5$, $\log n \leqslant o(|\Sigma|)$ in Theorem 18 we have*

$$\mathrm{E}[X] \leqslant \mathrm{E}[X^*] + o(1)\,.$$

The result of Corollary 19 is to be contrasted with previous work on practical implementations of linear probing. While Knuth's analysis serves as evidence of linear probing's efficieny in terms of the number of comparisons performed, the advantage of linear probing (over other hash table implementations) is that each sequential memory access is much faster than the random memory access we do for the first probe at $T[h(x)]$. How much faster depends on the computer system as does the cost of increasing the memory to reduce the load. Some experimental studies [7, 21, 34] have found linear probing to be the fastest hash table organization for moderate load factors (30-70%). If the load is higher, we could double the table size.

However, using experimental benchmarks to decide the hash table organization is only meaningful if the experiments are representative of the key sets on which linear probing would be employed. Since fully random hashing cannot be efficiently implemented, we might resort to weaker hash functions for which there could be inputs leading to much worse performance than in the benchmarks. The sensitivity to a bad choice of the hash function led [21] to advice *against* linear probing for general-purpose use. Indeed, Mitzenmacher and Vadhan [27] have proved that 2-independent hashing performs as well as fully random hashing if the input has enough entropy. However, [36, 30] have shown that with the standard 2-independent linear hashing scheme, if the input is a dense set (or more generally, a dense subset of an arithmetic sequence), then linear probing becomes extremely unreliable and the expected probe length increases from Knuth's $\frac{1+1/\varepsilon^2}{2}$ to $\Omega(\log n)$, while the best known upper bound in this case is $n^{o(1)}$ [24].[8]

In a breakthrough result, Pagh, Pagh and Ružić [29] showed that if we use 5-independent hashing and the load gap $\varepsilon = \Omega(1)$, then the expected probe length is constant. Pătraşcu and Thorup [31] generalized this to an $O(1/\varepsilon^2)$ bound for arbitrary ε, and showed that this also holds for simple tabulation hashing. However, in both cases, the analysis hides unspecified large constants in the O-notation. Thus, with these hashing schemes, there could still be inputs for which linear probing performs, say, 10 times worse in expectation, and then we would be better off using chaining.

Our result is of a very different nature. We show that whp, for any given query key x, the probe length corresponding to $h(x)$ when we use tornado tabulation hashing is stochastically dominated by the probe length in a linear probing table that hashes slightly more keys but uses fully random hashing. In particular, this implies that whp, the expected probe length with tornado tabulation hashing is only a factor $1 + o(1)$ away from Knuth's $\frac{1+1/\varepsilon^2}{2}$. We get this result without having to revisit Knuth's analysis from [25], but simply because we know that we are almost as good as fully random hashing, in a local sense that is sufficient for bounding the probe length (see Section 7.1).

As a further consequence of our results, we get that any benchmarking with random keys that we do in order to set system parameters will be representative for all possible sets of input keys.

[8]We note that if we only know that the hash function is 2-independent, then the lower bound for the expected probe length is $\Omega(\sqrt{n})$ and this is tight. [36, 30]

Moreover, the fact that tornado tabulation hashing only needs locality to perform almost as well as fully random hashing means that our arguments also work for other variants of linear probing. For instance, ones where the maintenance of the hash table prioritizes the keys depending on when they were inserted, as first suggested in [4]. Examples of this include Robin hood hashing where keys with the lowest hash value come first [8] or time-reversed linear probing [34] where the latest arrival comes first. In all these cases, tornado tabulation hashing performs almost as well as with fully-random hashing.

7.1 Proof of Theorem 18

We let $\alpha = n/m$ denote the fill of the hash table and $\varepsilon = 1 - \alpha$. The basic combinatorial measure that we study and employ is the *run length*: If cells $T[a]$, $T[a+1], \ldots, T[b-1]$ are all occupied by elements from S but both $T[a-1]$ and $T[b]$ are freem these $b-a$ cells are called a run of length $b-a$. Let $R(x, S)$ be the length of the run intersecting $T[h(x)]$ and note that $R(x, S)+1$ is an upper bound on the number of comparisons needed to insert some element y into the table when y hashes to the same location as x.

Let Δ be the largest power of two such that $3\alpha\Delta + 1 \leqslant |\Sigma|/2$. The following lemma, proven in Section 7.3, gives an upper bound on the probability that $h(x)$ intersects a long run.

Lemma 20.

$$\Pr[R(x, S) \geqslant \Delta] \leqslant \frac{1}{|\Sigma|} + 60 \log n \cdot \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma)\,.$$

Let $\mathcal{A}$ be the event $(R(x, S) \geqslant \Delta)$. Assuming $\mathcal{A}$, there exists at least one unoccupied cell in table T between $T[h(x)-\Delta]$ and $T[h(x)]$ and likewise between $T[h(x)]$ and $T[h(x)+\Delta]$. Hence the insertion of x only depends on the distribution of the much smaller key-set $\{s \in S \mid |h(s) - h(x)| \leqslant \Delta\}$.

The second step of our proof bounds the probability that tornado tabulation behaves like a fully random hash function when restricted to this small set of keys. As Theorem 5 doesn't apply for arbitrary intervals we will instead cover the necessary interval with three dyadic intervals. Recall that a dyadic interval is an interval of the form $[j2^i, (j+1)2^i)$, where i, j are integers. In the following we will exclusively consider a number of dyadic intervals all of length Δ. Let I_C denote the dyadic interval that contains $h(x)$, and similarly let I_R and I_L denote the dyadic intervals to the left and right, respectively, of I_C. We further let X_C be the set of keys in S that hash into the interval I_C, i.e., $X_C = \{x \in S \mid h(x) \in I_C\}$, and similarly, X_R and X_L are the pre-image of h in I_R and I_L, respectively. Given $\mathcal{A}$, the distribution of X is completely determined by the distribution of the keys in $X_L \cup X_C \cup X_R$ and $h(x)$.

The expected size of each preimage is $\alpha\Delta$ and our choice of Δ thus allows us to apply Theorem 5 to all three intervals at once. Let $\mathcal{B}$ be the event that the keys hashing into these intervals are distributed independently:

Corollary 21. *With probability at least* $1 - \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma)$*,* $\tilde{h}(X_R \cup X_C \cup X_L \cup \{x\})$ *is linearly independent, such that* h *hashes the keys in* $X_R \cup X_C \cup X_L \cup \{x\}$ *independently and uniformly in their respective intervals.*

We now define the analogous terms in the fully random setting. We let I_C^* denote the dyadic interval in T^* that contains $h^*(x)$ and I_R^* and I_L^* the right and left neighboring dyadic intervals. Similarly, we let X_C^*, X_R^* and X_L^* denote their preimages under h^*. The following lemma compares the two experiments in terms of the sizes of these preimages, and is proven in Section 7.2.

Lemma 22. *Let $\mathcal{C}$ be the event $|X_L| \leqslant |X_L^*| \wedge |X_C| \leqslant |X_C^*| \wedge |X_R| \leqslant |X_R^*|$, then*

$$\Pr\left[\mathcal{B} \wedge \bar{\mathcal{C}}\right] \leqslant 6\delta$$

Let $\mathcal{C}$ be the event that each of the preimages X_i contain at most as many elements as the corresponding preimage X_i^*. Finally, define $\mathcal{E} = \mathcal{A} \cap \mathcal{B} \cap \mathcal{C}$. We will now present a coupling $\tilde{X}$ of X which, when conditioned on $\mathcal{E}$, satisfies $\tilde{X} \leqslant X^*$.

For every realization of $|X_L|$, $|X_C|$ and $|X_R|$, we consider the following random process: starting from an emtpy table of size m and using the fully random h^*, insert the first $|X_L|$ elements from X_L^*, then the first $|X_R|$ elements from X_R^* and finally, the first $|X_C|$ elements from X_C^* (we do not insert any more elements after this). Note that, conditioned on $\mathcal{C}$, we have that it is possible to choose such elements (i.e., $|X_R| \leqslant |X_R^*|$ etc.). Now let $\tilde{X}$ denote the number of comparisons performed when inserting x into the table at this point in time.

We now have that X (defined for the tornado tabulation) is identically distributed as $\tilde{X}$ (defined for a fully random hash function). This is because event $\mathcal{A}$ implies that the distribution of X only depends on X_R, X_C and X_L. Event $\mathcal{B}$ further implies that, on these intervals, h behaves like a fully random hash function. Now note that $\tilde{X} \leqslant X^*$, since we can continue the random process and add the remaining keys in S^* and this can only increase the number of comparisons required to insert x (i.e., "more is worse").

Left is to compute the total probability that any of our required events fail:

$$\begin{aligned}
\Pr\left[\bar{\mathcal{E}}\right] &= \Pr\left[\bar{\mathcal{A}} \vee \bar{\mathcal{B}} \vee \bar{\mathcal{C}}\right] \\
&= \Pr\left[(\bar{\mathcal{A}} \vee \bar{\mathcal{C}}) \wedge \mathcal{B}\right] + \Pr\left[\bar{\mathcal{B}}\right] \\
&\leqslant \Pr\left[\bar{\mathcal{A}}\right] + \Pr\left[\mathcal{B} \wedge \bar{\mathcal{C}}\right] + \Pr\left[\bar{\mathcal{B}}\right] \\
&\leqslant 1/|\Sigma| + 6\delta + 61 \cdot \log n \cdot \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma)\,.
\end{aligned}$$

This concludes the proof.

7.2 Proof of Lemma 22

As Δ is chosen to be the largest power of two such that $3\alpha\Delta \leqslant |\Sigma|/2$ we get $\frac{|\Sigma|}{12\alpha} \leqslant \Delta$. Let t be a constant, to be decided later. For each $i \in \{L, C, R\}$ let $\mathcal{E}_i$ be the event $(|X_i| \leqslant t)$ and $\mathcal{E}_i^*$ be the event $(t \leqslant |X_i^*|)$.

$$\begin{aligned}
\Pr\left[\mathcal{B} \wedge \bar{\mathcal{C}}\right] &= \Pr[\mathcal{B} \wedge \exists i \in \{L, C, R\} : |X_i| > |X_i^*|] \\
&\leqslant \Pr\left[\mathcal{B} \wedge \exists i \in \{L, C, R\} : \bar{\mathcal{E}}_i \vee \bar{\mathcal{E}}_i^*\right] \\
&\leqslant \sum_{i \in \{L,C,R\}} (\Pr[\mathcal{B} \wedge |X_i| > t] + \Pr[\mathcal{B} \wedge |X_i^*| < t]) \\
&\leqslant \sum_{i \in \{L,C,R\}} (\Pr[\mathcal{B} \wedge |X_i| > t] + \Pr[|X_i^*| < t])
\end{aligned}$$

Let $\mu = \mathrm{E}[|X_i|] = \Delta\alpha$, $k = \sqrt{3\log(1/\delta)/\mu}$ and $t = (1 + k)\mu$. Applying the tail-bound of

Lemma 6 we have

$$\begin{aligned}\Pr[\mathcal{B} \wedge |X_i| > t] &= \Pr[\mathcal{B} \wedge |X_i| > (1+k)\mu] \\ &\leqslant \exp\left(-k^2\mu/3\right) \\ &= \delta\,.\end{aligned}$$

As $\mu = \alpha\Delta \geqslant |\Sigma|/12$ we have

$$\begin{aligned}k &= \sqrt{3\log(1/\delta)/(\alpha\Delta)} \\ &\leqslant \sqrt{36\log(1/\delta)/|\Sigma|} \\ &\leqslant 1/3\,.\end{aligned}$$

As $n^* = (1+15\sqrt{\log(1/\delta)/|\Sigma|})n \geqslant (1+2.5k)n$, $\mu^* = \mathrm{E}[|X_i^*|] \geqslant (1+2.5k)\mu$. Next, let $k^* = k\cdot\sqrt{2/3}$. Then

$$\begin{aligned}\mu^* \cdot (1-k^*) &\geqslant (1+2.5k)\cdot\mu\cdot(1-k^*) \\ &= (1+2.5k)\cdot\mu\cdot(1-\sqrt{2/3}\cdot k) \\ &\geqslant \mu\cdot(1+k) \\ &= t\,.\end{aligned}$$

Hence

$$\begin{aligned}\Pr[|X_i^*| < t] &\leqslant \Pr[|X_i^*| < (1-k^*)\mu^*] \\ &\leqslant \exp(-(k^*)^2\mu^*/2) \\ &= \exp(-k^2\mu^*/3) \\ &\leqslant \exp(-k^2\mu/3) \\ &= \delta\,.\end{aligned}$$

Summing over the six cases we see $\Pr[\mathcal{B} \wedge \exists i \in \{L, C, R\} : |X_i| > |X_i^*|] \leqslant 6\delta$.

7.3 Proof of Lemma 20

Our proof relies on the simple observation that if $T[a]$ through $T[b]$ are all occupied and the run *starts* in $T[a]$ (i.e. $T[a-1]$ is free which excludes the possibility of prior positions spilling over), then the preimage $h^{-1}([a,b]) = \{s \in S \mid h(x) \in [a,b]\}$ must have size at least $|b-a|$. It must also be the case that either (1) the preimage $h^{-1}([a+1,b])$ has size at least $(b-a)\cdot(1-\gamma)$ or (2) the preimage $h^{-1}([a,a])$ is of size at least $(b-a)\cdot\gamma$, for any parameter $\gamma \geqslant 0$.

We can generalize this to consider a run starting in any position $T[b]$ within some interval $b \in [a,c]$ which continues through $T[d]$, then either (1) $\left|h^{-1}([c,d])\right| \geqslant (d-c)\cdot(1-\gamma)$ or (2) $\left|h^{-1}([a,c])\right| \geqslant (d-c)\cdot\gamma$. We will refer to $[a,c]$ as the start-interval and to $[c,d]$ as the long interval.

Our strategy is to make both of these events unlikely by balancing the size of the start-interval with the number of keys needed to fill up the long interval. Larger difference $(d-c)$ allows for a larger start-interval. With a collection of roughly $\log_{1+\varepsilon/(6\alpha)} m$ such start-intervals we cover all possible starting poisitions before $T[h(x)-\Delta]$, ruling out the possibility that a run starting before

$T[h(x) - \Delta]$ reaches $T[h(x)]$. The same strategy applied once more rules out the possibility that the run intersecting $T[h(x)]$ will continue through $T[h(x) + \Delta]$.

For our proof we set $\gamma = \varepsilon/3$ where $\varepsilon = 1 - n/m$ is the fill gap of T. The first pair of intervals we consider is the long interval $I_0 = [h(x) - \Delta, h(x)]$ and the start-interval I_0' of size $\Delta\varepsilon/(6\alpha)$ preceding I_0. Next follows the long interval $I_1 = I_0' \cup I_0$ with start-interval I_1' of length $|I_1|\,\varepsilon/(6\alpha)$ preceding it, and so forth. Let $\Delta_i = |I_i|$ and $\Delta_i' = |I_i'|$. Observe how $\Delta_i = \Delta \cdot (1 + \varepsilon/(6\alpha))^i$, bounding the number of needed interval-pairs at $\log_{1+\varepsilon/(6\alpha)}(m/\Delta)$.

Depending on the length of the interval being inspected we can apply either Lemma 6 or Lemma 17 to bound the probability that the preimage exceeds the given threshold. Taking the maximum of the two bounds simplifies the analysis. Letting X be the size of a preimage, $\mu = \mathrm{E}[X]$ and $\delta \leqslant 1$ we obtain

$$\Pr[X \geqslant (1 + \delta)\mu] \leqslant \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma) + 4 \cdot \exp\left(-\delta^2 \cdot \min\{|\Sigma|/2, \mu\}/3\right) .$$

Let X_i be the size of the preimage of the long interval I_i of length Δ_i with $\mu_i = \alpha\Delta_i$. Then

$$\begin{aligned}\Pr[X_i \geqslant (1 - \varepsilon/3)\Delta_i] &= \Pr\left[X_i \geqslant \left(1 + \frac{2\varepsilon}{3\alpha}\right)\alpha\Delta_i\right] \\ &\leqslant \Pr\left[X_i \geqslant \left(1 + \frac{2\varepsilon}{3}\right)\mu_i\right] \\ &\leqslant \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma) + 4 \cdot \exp\left(-\left(\frac{2\varepsilon}{3}\right)^2 \cdot \frac{\min\{\mu_i, |\Sigma|/2\}}{3}\right)\end{aligned}$$

Notice how the probability is non-increasing for increasing sizes of the intervals. Thus we bound each of the probabilites for a long interval exceeding its threshold by the probability obtained for I_0 with $\mu_0 = \alpha\Delta \leqslant |\Sigma|/2$.

For start-interval I_i' of length Δ_i' with $\mu_i' = \alpha\Delta_i' = \Delta_i \cdot \varepsilon/6$ we observe the same pattern

$$\begin{aligned}\Pr\left[X_i' \geqslant \varepsilon/3\Delta_i\right] &= \Pr\left[X_i' \geqslant 2\mu_i'\right] \\ &\leqslant \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma) + 4 \cdot \exp\left(-\frac{\max\{\mu_i', |\Sigma|/2\}}{3}\right) ,\end{aligned}$$

where we can bound the probability that each start-interval is too large by the probability obtained for I_0' with $\mu_0' = \Delta\varepsilon/6$.

The probability that *any* of our intervals is too large is thus at most

$$2\log_{1+\varepsilon/(6\alpha)} m \cdot \left(\mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma) + 4\exp\left(-4/27 \cdot \varepsilon^2\alpha\Delta\right)\right)$$

where use that $4/9 \cdot \varepsilon^2\alpha\Delta \leqslant \varepsilon\Delta/9 \leqslant \varepsilon\Delta/6$ as $\varepsilon + \alpha = 1$, hence the probability obtained for I_0 is larger than that for I_0'.

Let us rewrite this expression in terms of n and $|\Sigma|$, our main parameters.

$$\begin{aligned}\log_{1+\varepsilon/(6\alpha)} m &= \log_{1+\varepsilon/(6\alpha)} n + \log_{1+\varepsilon/(6\alpha)}(1/\alpha) \\ &\leqslant \log n \cdot \log_{1+\varepsilon/(6\alpha)}(2) + 6 \\ &\leqslant \log n \cdot 6\alpha/\varepsilon + 6 \\ &\leqslant 30 \cdot \log n ,\end{aligned}$$

using $\varepsilon \geqslant 1/5$. As $\alpha\Delta \geqslant |\Sigma|/12$, we get

$$\begin{aligned}4\exp\left(-4/27 \cdot \varepsilon^2\alpha\Delta\right) &\leqslant 4\exp\left(-4/27 \cdot \varepsilon^2|\Sigma|/12\right)\\ &\leqslant 4\exp\left(-1/2025 \cdot |\Sigma|\right)\\ &\leqslant \frac{1}{2|\Sigma|^2}\end{aligned}$$

as $|\Sigma| \geqslant 2^{16}$. Assuming $30 \cdot \log n \leqslant |\Sigma|$ the total error-probability becomes

$$1/(2|\Sigma|) + 30 \cdot \log n \cdot \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma)\,.$$

Repeating the process once more to ensure that the run at $T[h(x)]$ doesn't continue past $T[h(x)+\Delta]$ doubles the error-probability and proves the lemma.

7.4 Proof of Corollary 19

To bound the expected number of comparisons we rely on the following lemma from [31] which gives strong concentration bounds for the runlength when applied to our simple tabulation $\widehat{h}$.

Lemma 23 (Corollary 3.2 in [31]). *For any $\gamma = O(1)$ and $\ell \leqslant n^{1/(3(c+d))}/\alpha$,*

$$\Pr[R(x,S) \geqslant \ell] \leqslant \begin{cases} 2e^{-\Omega(\ell\varepsilon^2)} + (\ell/m)^\gamma & \text{if } \alpha \geqslant 1/2\\ \alpha^{\Omega(\ell)} + (\ell/m)^\gamma & \text{if } \alpha \leqslant 1/2\end{cases}$$

where the constants hidden in O and Ω are functions of $c+d$, the size of the derived keys on which we apply simple tabulation.

In particular this implies that, for some $\ell = \Theta\left((\log n)/\varepsilon^2\right)$, we have that $R(q,S) \geqslant \ell$ with probability at most $1/n^{10}$. Let $\mathcal{E}$ be the event of stochastic dominance, as given by Theorem 18, and $\mathcal{A}$ the event $(R(x,S) \leqslant \ell)$. Then

$$\mathrm{E}[X] = \mathrm{E}[X \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] + \mathrm{E}\big[X \mid \bar{\mathcal{E}} \wedge \mathcal{A}\big] \cdot \Pr\big[\bar{\mathcal{E}} \wedge \mathcal{A}\big] + \mathrm{E}\big[X \mid \bar{\mathcal{E}} \wedge \bar{\mathcal{A}}\big] \cdot \Pr\big[\bar{\mathcal{E}} \wedge \bar{\mathcal{A}}\big]\,.$$

First, observe

$$\begin{aligned}\mathrm{E}[X \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] &= \sum_{i=1} \Pr[X \geqslant i \mid \mathcal{E}] \cdot \Pr[\mathcal{E}]\\ &\leqslant \sum_{i=1} \Pr[X^* \geqslant i \mid \mathcal{E}] \cdot \Pr[\mathcal{E}]\\ &= \sum_{i=1} \Pr[X^* \geqslant i \wedge \mathcal{E}]\\ &\leqslant \sum_{i=1} \Pr[X^* \geqslant i]\\ &= \mathrm{E}[X^*]\,.\end{aligned}$$

With $\delta = 1/|\Sigma|$ and $d \geqslant 5$, $\Pr\big[\bar{\mathcal{E}}\big] \leqslant 9/|\Sigma|$. Assuming $\mathcal{A}$, the next open cell of T is at most ℓ positions away,

$$\begin{aligned}\mathrm{E}\big[X \mid \bar{\mathcal{E}} \wedge \mathcal{A}\big] \cdot \Pr\big[\bar{\mathcal{E}} \wedge \mathcal{A}\big] &\leqslant \ell \cdot \Pr\big[\bar{\mathcal{E}}\big]\\ &\leqslant \Theta\left(\frac{\log n}{\varepsilon^2}\right) \cdot \frac{9}{|\Sigma|}\\ &\leqslant o(1)\,.\end{aligned}$$

Finally, no more than n comparisons will ever be necessary. Hence

$$\begin{aligned} \mathrm{E}\big[X \mid \bar{\mathcal{E}} \wedge \bar{\mathcal{A}}\big] \cdot \Pr\big[\bar{\mathcal{E}} \wedge \bar{\mathcal{A}}\big] &\leqslant n \cdot \Pr\big[\bar{\mathcal{A}}\big] \\ &\leqslant 1/n^9 \,. \end{aligned}$$

This gives the desired bound on $\mathrm{E}[X]$,

$$\mathrm{E}[X] \leqslant \mathrm{E}[X^*] + o(1) + 1/n^9 \,.$$

8 Lower Bound for Tornado Tabulation

In this section, we show that the probability obtained in Theorem 4 is tight up to constant factors. Specifically, we will prove the following:

Theorem 7. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters. There exists a selector function f with $\mu^f \leqslant \Sigma/2$ such that the derived selected keys $\widetilde{h}(X^{f,h})$ are linearly dependent with probability at least $\Omega((3/|\Sigma|)^{d-2})$.*

Our strategy will mimic that in the proof of Theorem 4 and show that the set of derived selected keys will contain a zero-set with probability at least $\Theta((\mu^f)^3(3/|\Sigma|)^{d+1})$. We begin by establishing some initial general bounds. In the following, we define $\widetilde{h}' : \Sigma^c \to \Sigma^{c+d}$ to map keys in Σ^c to *simple* derived keys in Σ^{c+d} by applying the same functions as $\widetilde{h}$ except with $\widetilde{h}_0(\cdot) = 0$, i.e., for all $i > 1$, $\widetilde{h}'_{c+i} = \widetilde{h}_{c+i}$.

Definition 1. We say that a zero-set $Y \subseteq \Sigma^c$ *survives* d rounds of tornado tabulation if the set $\widetilde{h}'(Y) \subseteq \Sigma^{c+d}$ of its simple derived keys is also a zero-set.

We focus on zero-sets of size 4 and lower bound the probability that they survive successive rounds of tornado tabulation. We first define some terminology necessary to describe how each new derived character in $\widetilde{x}'$ behaves. Specifically, let $Y = \{x_1, x_2, x_3, x_4\}$ be a zero-set, for some fixed ordering of its keys. We distinguish between four types of positions $i \in \{1, \dots, c\}$ as such: (1) position i is of Type A iff $x_1[i] = x_2[i]$ and $x_3[i] = x_4[i]$, (2) it is of Type B iff $x_1[i] = x_3[i]$, $x_2[i] = x_4[i]$, (3) it is of Type C iff $x_1[i] = x_4[i]$ and $x_2[i] = x_3[i]$ and, (4) it is of Type D iff $x_1[i] = x_2[i] = x_3[i] = x_4[i]$. We now prove that"

Lemma 24. *Let $Y \subseteq \Sigma^c$ be a zero-set with $|Y| = 4$. Then, for any $c \geqslant 2$, Y survives one round of tornado tabulation with probability* $(3 - 2/|\Sigma|)\,/\,|\Sigma|$.

Proof. Since the original keys in Y already form a zero-set, the set of simple derived keys $\widetilde{h}'(Y)$ is a zero-set iff the set of simple derived characters $\widetilde{h}'_{c+1}(Y)$ is a zero-set. Moreover, the cases in which $\widetilde{h}'_{c+1}(Y)$ is a zero-set can be classified based on the type of position $c+1$. Specifically, let $\mathcal{A}_{c+1}$ denote the event that position $c+1$ is of Type A, i.e., $\widetilde{h}'_{c+1}(x_1) = \widetilde{h}'_{c+1}(x_2)$ and $\widetilde{h}'_{c+1}(x_3) = \widetilde{h}'_{c+1}(x_4)$, and similarly for $\mathcal{B}_{c+1}$, $\mathcal{C}_{c+1}$, and $\mathcal{D}_{c+1}$. Then

$$\begin{aligned} \Pr\left(Y \text{ survives one round}\right) &= \Pr\left(\widetilde{h}'_{c+1}(Y) \text{ is a zero-set }\right) \\ &= \Pr\left(\mathcal{A}_{c+1} \vee \mathcal{B}_{c+1} \vee \mathcal{C}_{c+1}\right) \\ &= \Pr\left(\mathcal{A}_{c+1}\right) + \Pr\left(\mathcal{B}_{c+1}\right) + \Pr\left(\mathcal{C}_{c+1}\right) - \Pr\left(\mathcal{A}_{c+1} \wedge \mathcal{B}_{c+1}\right) - \\ &\Pr\left(\mathcal{A}_{c+1} \wedge \mathcal{C}_{c+1}\right) - \Pr\left(\mathcal{B}_{c+1} \wedge \mathcal{C}_{c+1}\right) + \Pr\left(\mathcal{A}_{c+1} \wedge \mathcal{B}_{c+1} \wedge \mathcal{C}_{c+1}\right) \\ &= 3\Pr\left(\mathcal{A}_{c+1}\right) - 2\Pr\left(\mathcal{D}_{c+1}\right) \ , \end{aligned}$$

where the last equality follows from the fact that the events $\mathcal{A}_{c+1}$, $\mathcal{B}_{c+1}$ and $\mathcal{C}_{c+1}$ are equivalent up to a permutation of the elements in Y, and the fact that the conjunction of any pair of events in $\mathcal{A}_{c+1}$, $\mathcal{B}_{c+1}$ and $\mathcal{C}_{c+1}$ implies $\mathcal{D}_{c+1}$, and vice-versa.

We now bound $\Pr(\mathcal{A}_{c+1})$ and $\Pr(\mathcal{D}_{c+1})$. Recall that, by definition, the simple derived character $\widetilde{h}'_{c+1}(x)$ is the output of a simple tabulation hash function applied to the key x. Specifically, for each $i \in \{1, \ldots, c\}$, let $T_i : \Sigma \to \Sigma$ denote a fully random function. Then

$$\widetilde{h}'_{c+1}(x) = T_1(x[1]) \oplus \ldots \oplus T_c(x[c]) \ .$$

Let I_A, I_B, I_C and I_D partition the set of positions in the original keys $\{1, \ldots, c\}$ based on their type, i.e., I_A consists of positions that are of Type A but not Type D, similarly for I_B and I_C, and finally I_D denotes the positions of Type D. We then define $T_A(x) = \oplus_{i \in I_A} T_i[x[i]]$ and similarly $T_B(x)$, $T_C(x)$, and $T_D(x)$. When $\mathcal{A}_{c+1}$ happens, we have that $\widetilde{h}'_{c+1}(x_1) = \widetilde{h}'_{c+1}(x_2)$, which is equivalent to

$$T_B(x_1) \oplus T_C(x_1) = T_B(x_2) \oplus T_C(x_2) \ ,$$

since $T_A(x_1) = T_A(x_2)$ and $T_D(x_1) = T_D(x_2)$ by definition. Similarly, $\widetilde{h}'_{c+1}(x_3) = \widetilde{h}'_{c+1}(x_4)$, is equivalent to

$$T_B(x_3) \oplus T_C(x_3) = T_B(x_4) \oplus T_C(x_4) \ .$$

Note that, by definition, $x_1[i] = x_3[i]$ and $x_2[i] = x_4[i]$ for all $i \in I_B$, and hence $T_B(x_1) = T_B(x_3)$ and $T_B(x_2) = T_B(x_4)$. Similarly, $T_C(x_1) = T_C(x_4)$ and $T_C(x_2) = T_C(x_3)$. Therefore, both equalities are equivalent to

$$T_B(x_1) \oplus T_C(x_1) \oplus T_B(x_2) \oplus T_C(x_2) = 0 \ .$$

Given that $x_1[i] \neq x_2[i]$ for all $i \in I_B \cup I_C$ and the T_i's are independent, we have that

$$\Pr[T_B(x_1) \oplus T_C(x_1) \oplus T_B(x_2) \oplus T_C(x_2) = 0] = 1/|\Sigma| \ .$$

In order to bound $\Pr(\mathcal{D}_{c+1})$, we first note that

$$\Pr(\mathcal{D}_{c+1}) = \Pr(\mathcal{A}_{c+1} \wedge \mathcal{B}_{c+1}) = \Pr(\mathcal{A}_{c+1}) \cdot \Pr(\mathcal{B}_{c+1} \mid \mathcal{A}_{c+1}) = 1/|\Sigma| \cdot \Pr(\mathcal{B}_{c+1} \mid \mathcal{A}_{c+1}) \ .$$

A similar argument as before shows that event $\mathcal{B}_{c+1}$ is equivalent to

$$T_A(x_1) \oplus T_C(x_1) \oplus T_A(x_3) \oplus T_C(x_3) = 0 \ .$$

Note, in particular, that the event $\mathcal{B}_{c+1}$ depends on positions in I_A and I_C, while the event $\mathcal{A}_{c+1}$ depends on positions in I_B and I_C. Moreover, it cannot be that both I_A and I_B are empty, since then we would not have a zero-set of size 4 (i.e., we would get that $x_1 = x_4$ and $x_2 = x_3$). Therefore, $I_B \cup I_C \neq I_A \cup I_C$ and the two events $\mathcal{A}_{c+1}$ and $\mathcal{B}_{c+1}$ are independent, and so $\Pr(\mathcal{D}_{c+1}) = 1/|\Sigma|^2$. The claim follows. $\square$

As a corollary, we get the following:

Corollary 25. *For any $c \geqslant 2$, a zero-set $Y \subseteq \Sigma^c$ with $|Y| = 4$ survives d rounds of tornado tabulation with probability* $\left((3 - 2/|\Sigma|)/|\Sigma|\right)^d$.

Proof. We prove the claim by induction on d and note that the case in which $d = 1$ is covered in Lemma 24. Now assume that the statement is true for $d-1$. Recall that $\widetilde{x}'$ denotes the simple derived key and that $\widetilde{x}'[\leqslant c+d-1]$ denotes the first $c+d-1$ characters of $\widetilde{x}'$. By extension, let $\widetilde{Y}'$ denote the set of simple derived keys of Y and similarly, $\widetilde{Y}'[\leqslant c+d-1] = \{\widetilde{x}'[\leqslant c+d-1] \mid x \in Y\}$ and $\widetilde{Y}'[c+d] = \{\widetilde{x}'[c+d] \mid x \in Y\}$. Finally, let $\mathcal{E}_{d-1}$ and $\mathcal{E}_d$ denote the events that the set $\widetilde{Y}'$ and $\widetilde{Y}'[\leqslant c+d-1]$, respectively, are zero-sets. Then:

$$\begin{aligned}\Pr(\mathcal{E}_d) &= \Pr\left(\mathcal{E}_{d-1} \text{ and } \widetilde{Y}'[c+d] \text{ is a zero-set}\right)\\ &= \Pr(\mathcal{E}_{d-1}) \cdot \Pr\left(\widetilde{Y}'[c+d] \text{ is a zero-set } \mid \mathcal{E}_{d-1}\right)\\ &= ((3-2/|\Sigma|)/|\Sigma|)^{d-1} \cdot \Pr\left(\widetilde{Y}'[c+d] \text{ is a zero-set } \mid \mathcal{E}_{d-1}\right) ,\end{aligned}$$

where the last equality holds by the inductive hypothesis. To finish things up, we note that the event $\widetilde{Y}'[c+d]$ conditioned on $\mathcal{E}_{d-1}$ is equivalent to the set $\widetilde{Y}'[\leqslant c+d-1]$ surviving one round of tabulation hashing. Hence, it happens with probability $(3-2/|\Sigma|)/|\Sigma|$ and the claim follows.

□

8.1 Proof of Theorem 7

The hard instance. Consider the set of keys $S = \{0,1\} \times \Sigma$ and note that $\widetilde{h}_0$ on this set induces a permutation of the characters in Σ. Specifically, every key of the form $0c$ for some $c \in \Sigma$ will be mapped to the element $0c'$, where $c' = \widetilde{h}_0(0) \oplus c$ and the mapping $c \to \widetilde{h}_0(0) \oplus c$ is a permutation. Similarly for keys $1c$. Therefore, we can assume without loss of generality that $\widetilde{h}_0(\cdot) = 0$ and get that:

$$\Pr\left(\widetilde{h}(X^{f,h}) \text{ is linearly dep.}\right) = \Pr\left(\exists \text{ a four-set } Y \subseteq X^{f,h} \text{ that survives } d \text{ rounds of tornado tab.}\right).$$

We then define the selector function to select a key x if $x \in S$ and the two leftmost output characters of $h(x)$ are both 0. Note then than that the probability that an $x \in S$ gets selected to $X^{f,h}$ is $1/4$ and hence, $\mu^f = |\Sigma/2|$. We now focus on zero-sets from S of size 4 and, for any $c_1, c_2 \in \Sigma$ with $c_1 < c_2$, we denote the zero-set of size four $\{0c_1, 1c_1, 0c_2, 1c_2\}$ by $Y(c_1, c_2)$. We then let $\mathcal{Y} = \{Y(c_1,c_2) \mid c_1 < c_2 \in \Sigma\}$ and let $\mathcal{E}_i(Y)$ denote the event that a zero-set Y survives i rounds of tornado tabulation. We lower bound the probability that $\widetilde{h}(X^{f,h})$ is linearly dependent by only focusing on zero-sets in $\mathcal{Y}$:

$$\begin{aligned}\Pr\left(\widetilde{h}(X^{f,h}) \text{ is linearly dep.}\right) &\geqslant \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_d(Y) \wedge Y \subseteq X^{f,h}\right)\\ &\geqslant \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_d(Y) \wedge Y \subseteq X^{f,h}\right)\\ &\geqslant \Pr(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_d(Y)) \cdot \Pr\left(\text{a fixed } Y \in \mathcal{Y}, Y \subseteq X^{f,h} \;\middle|\; \mathcal{E}_d(Y)\right)\\ &\geqslant 1/4^3 \cdot \Pr(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_d(Y)) ,\end{aligned}$$

where the last two inequalities above are due to the fact that the probability that some fixed set $Y \in \mathcal{Y}$ gets selected in $X^{f,h}$ given that it survived d rounds of tabulation hashing is the same across all sets in $\mathcal{Y}$ and, furthermore, it is exactly $1/4^3$.

Surviving zero-sets. We now employ Corollary 25 to lower bound the probability that some zero-set in $\mathcal{Y}$ survives d rounds of tornado tabulation. Note that we already have that the expected number of zero-sets in $\mathcal{Y}$ that survive d rounds of tornado tabulation is

$$\Theta(|\Sigma|^2 \cdot ((3 - 2/|\Sigma|) / |\Sigma|)^d) = \Omega((3/|\Sigma|)^{d-2}) ,$$

which exhibits the desired dependency on d. The challenge with turning this expectation into a probability is that the events of sets in $\mathcal{Y}$ surviving a round are not independent. To address this, we decompose the event of some set Y surviving d rounds of tornado tabulation into the event that some set survives the first two rounds of tornado tabulation and the event that this set also survives the remaining $d - 2$ rounds:

$$\begin{aligned}
\Pr\left(\widetilde{h}(X^{f,h}) \text{ is linearly dep.}\right) &\geqslant 1/4^3 \cdot \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_d(Y)\right) \\
&\geqslant 1/4^3 \cdot \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_2(Y)\right) \cdot \Pr\left(\mathcal{E}_{d-2}(\widetilde{h}(Y)[\leqslant c+2]) \;\middle|\; \mathcal{E}_2(Y)\right) \\
&= 1/4^3 \cdot ((3 - 2/|\Sigma|) / |\Sigma|)^{d-2} \cdot \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_2(Y)\right) .
\end{aligned}$$

To finish the argument and prove the main claim, we show the following:

Lemma 26. *With constant probability, at least one set in $\mathcal{Y}$ survives the first two rounds of tornado tabulation.*

Proof. The proof proceeds in two stages: first, we will argue that, with constant probability, $\Theta(|\Sigma|)$ of the sets in $\mathcal{Y}$ survive the first round of tornado tabulation. These sets will have a specific structure that guarantees that they then survive the second round of tornado tabulation independently.

Let $T_1^{(1)}, T_2^{(1)} : \Sigma \to \Sigma$ be the two fully random hash functions involved in computing the first derived character, and let $\mathcal{C}_1$ denote the event that $T_1^{(1)}(0) \neq T_1^{(1)}(1)$. Note that $\mathcal{C}_1$ happens with probability $1 - 1/|\Sigma|$. Conditioned on $\mathcal{C}_1$, all the sets in $\mathcal{Y}$ that survive have position 3 of Type B or C. For any $Y(c_1, c_2) \in \mathcal{Y}$, position 3 is of Type B if $T_2^{(1)}(c_1) = T_2^{(1)}(c_2)$ and of Type C if $T_2^{(1)}(c_2) = T_1^{(1)}(0) \oplus T_1^{(1)}(1) \oplus T_2^{(1)}(c_1)$. These events are mutually exclusive and each occurs with probability $1/|\Sigma|$.

We now show that, with constant probability, at least $\Theta(|\Sigma)|$ of sets in $\mathcal{Y}$ will survive and futher, have position 3 be of Type B. We model this as a balls-into-bins game in which there is a bin for each character $\alpha \in \Sigma$ and the characters in Σ hash into bins using $T_2^{(1)}$. We let N_α denote the number of characters $c \in \Sigma$ with $T_2^{(1)}(c_1) = \alpha$, i.e., the occupancy of the bin for α. We are interested in events in which $N_\alpha \geqslant 2$, because this implies that there exists at least one set $Y(c_1, c_2) \in \mathcal{Y}$ where $T_2^{(1)}(c_1) = T_2^{(1)}(c_2) = \alpha$. The probability that this occurs is:

$$\Pr(N_\alpha \geqslant 2) = 1 - \Pr(N_\alpha = 0) - \Pr(N_\alpha = 1) = 1 - (1 - 1/|\Sigma|)^{|\Sigma|} - (1 - 1/|\Sigma|)^{|\Sigma|-1} ,$$

since each character hashes independently and uniformly into the bins. Now, for each $\alpha \in \Sigma$, define I_α to be the indicator random variable for whether $N_\alpha \geqslant 2$ and let $I = \sum_{\alpha \in \Sigma} I_\alpha$. We will argue that $I = \Theta(\Sigma)$ with constant probability. First note that the random variables $\{I_\alpha\}_{\alpha \in \Sigma}$ are negatively associated since bin occupancies are negatively associated [17]. Let $\mu = \mathrm{E}(I)$ and note that $\mu \geqslant |\Sigma|/4$ for $|\Sigma| \geqslant 2$. As such, we can apply Chernoff's bound and get that:

$$\Pr(I \leqslant |\Sigma|/8) \leqslant \Pr(I \leqslant (1-1/2)\cdot\mu) \leqslant e^{-\mu/8} \leqslant 1/2\ ,$$

where the last inequality holds when $|\Sigma| \geqslant 23$.

Now let $\mathcal{Y}' \subset \mathcal{Y}$ denote the set of zero-sets constructed as such: we partition the bins into subsets of the form $\{\alpha, \alpha'\}$ where $\alpha \oplus \alpha' = T_1^{(1)}(0) \oplus T_1^{(1)}(1)$. The event that $I \geqslant |\Sigma|/8$ implies that there are at least $|\Sigma|/16$ such subsets where at least one bin, say α, has $N_\alpha \geqslant 2$. Now let $c_1 < c_2$ be two such characters that hash into the bin and add the zero-set $Y(c_1, c_2)$ to $\mathcal{Y}'$. Note that every subset of bins contributes at most one zero-set to $\mathcal{Y}'$. Furthermore, the sets in $\mathcal{Y}'$ have the following properties:

- for any two distinct sets $Y(c_1, c_2), Y(c_3, c_4) \in \mathcal{Y}'$, it holds that $\{c_1, c_2\} \cap \{c_3, c_4\} = \varnothing$, since the characters c_1, c_2 hash into a different bin from c_3, c_4,
- if we denote the derived characters of $Y(c_1, c_2)$ by $\widetilde{h}_3(0c_1) = x_1$ and $\widetilde{h}_3(1c_1) = x_2$ and similarly, the derived characters of $Y(c_3, c_4)$ by $\widetilde{h}_3(0c_3) = y_1$ and $\widetilde{h}_3(1c_3) = y_2$, we have further that $\{x_1, x_2\} \cap \{y_1, y_2\} = \varnothing$. This is due to the way the derived characters are computed: on one hand, $x_1 = T_1^{(1)}(0) \oplus T_2^{(1)}(c_1) \neq T_1^{(1)}(0) \oplus T_2^{(1)}(c_3) = y_1$ because $T_2^{(1)}(c_1) \neq T_2^{(1)}(c_3)$ and similarly for $x_2 \neq y_2$. On the other hand, $x_1 \neq y_2$ because otherwise we would get that $T_2^{(1)}(c_1) \oplus T_2^{(1)}(c_3) = T_1^{(1)}(0) \oplus T_1^{(1)}(1)$, which would contradict the fact that $Y(c_1, c_2)$ and $Y(c_3, c_4)$ were generated by different subsets of bins.

In this context, the events in which sets in $\mathcal{Y}'$ survive the second round of tornado tabulation are independent. Specifically, let $Y(c_1, c_2) \in \mathcal{Y}'$ be a set as before with derived characters $\widetilde{h}_3(0c_1) = \widetilde{h}_3(0c_2) = x_1$ and $\widetilde{h}_3(1c_1) = \widetilde{h}_3(1c_2) = x_2$. We distinguish between whether the newly derived character is of Type A, B, or C. To this end, let $T_1^{(2)}, T_2^{(2)}, T_3^{(2)} : \Sigma \to \Sigma$ be the fully random hash functions involved in its computation. Then position 4 is of Type A if

$$T_1^{(2)}(0) \oplus T_3^{(2)}(x_1) = T_1^{(2)}(1) \oplus T_3^{(2)}(x_2)\ . \tag{24}$$

Position 4 is of Type B if

$$T_2^{(2)}(c_1) = T_2^{(2)}(c_2)\ , \tag{25}$$

and of Type C if

$$T_1^{(2)}(0) \oplus T_2^{(2)}(c_1) \oplus T_3^{(2)}(x_1) = T_1^{(2)}(1) \oplus T_2^{(2)}(c_2) \oplus T_3^{(2)}(x_2)\ . \tag{26}$$

Similar conditions hold for some other $Y(c_3, c_4) \in \mathcal{Y}'$, and moreover, each of them depends on values that are chosen independently from the values in $Y(c_1, c_2)$. Specifically, the analogues of Equation (24) for $Y(c_3, c_4)$ depends on the lookup table values of y_1 and y_2, where y_1 and y_2 are the derived characters $\widetilde{h}_3(0c_3) = y_1$ and $\widetilde{h}_3(1c_3) = y_2$, respectively. As noted before, we know that $\{x_1, x_2\} \cap \{y_1, y_2\} = \varnothing$, and so the analogue of Equation (24) for $Y(c_3, c_4)$ is independent of Equations (24), (25), and (26). Similar arguments can be made for the other cases.

Now fix some instantiation Y' of $\mathcal{Y}'$ of size $|\Sigma|/16$ and let $X_{Y'}$ denote the number of sets in Y' that survive the second round of tornado tabulation. We know from Lemma 24 that each set in

Y' survives the second round of tornado tabulation with probability $(3 - 2/|\Sigma|)/|\Sigma| \geqslant 2.75/16$ for $|\Sigma| \geqslant 8$. Furthermore, each set survives this second round independently from the others. It follows from Chernoff's inequality that X is then tightly concentrated around its mean. In particular,

$$\begin{aligned}\Pr\left(X_{Y'} \leqslant 0.01 \cdot 1/8 \,\middle|\, \mathcal{C}_1\right) &\leqslant \Pr\left(X_{Y'} \leqslant (1-0.99)\cdot \mathrm{E}[X_{\mathcal{Y}'}] \,\middle|\, \mathcal{C}_1\right) \\ &\leqslant e^{-\mathrm{E}[X_{\mathcal{Y}'}]\cdot 0.99^2/2} \leqslant e^{-2.75\cdot 0.99^2/32} \leqslant e^{-0.08} .\end{aligned}$$

For our purposes, let $X_{\mathcal{Y}'}$ denote the random variable that counts the number of sets in $\mathcal{Y}'$ that survive the second round of tabulation hashing. Conditioned on the fact that $|\mathcal{Y}'| \geqslant |\Sigma|/16$, we can always pick an instantiation Y' of $\mathcal{Y}'$ on which to use the above bound. We then get that:

$$\Pr\left(X_{\mathcal{Y}'} \geqslant 0.01/8 \,\middle|\, |\mathcal{Y}'| \geqslant |\Sigma|/16 \wedge \mathcal{C}_1\right) \geqslant 1 - e^{-0.08} .$$

To put it all together, let $I_{\mathcal{Y}}$ denote the event that there exists $Y(c_1, c_2) \in \mathcal{Y}$ such that $Z_2(c_1, c_2)$. Recall that we defined the event $\mathcal{C}_1$ to be that $T_1^{(1)}(0) \neq T_1^{(1)}(1)$. Then:

$$\begin{aligned}\Pr(I_{\mathcal{Y}}) &\geqslant \Pr(I_{\mathcal{Y}} \wedge \mathcal{C}_1) \\ &= \left(1 - \frac{1}{|\Sigma|}\right)\cdot \Pr\left(I_{\mathcal{Y}} \,\middle|\, \mathcal{C}_1\right) \\ &\geqslant \left(1 - \frac{1}{|\Sigma|}\right)\cdot \Pr\left(I_{\mathcal{Y}} \wedge |\mathcal{Y}'| \geqslant |\Sigma|/16 \,\middle|\, \mathcal{C}_1\right) \\ &\geqslant \left(1 - \frac{1}{|\Sigma|}\right)\cdot \frac{1}{2}\cdot \Pr\left(I_{\mathcal{Y}} \,\middle|\, |\mathcal{Y}'| \geqslant |\Sigma|/16 \wedge \mathcal{C}_1\right) \\ &\geqslant \left(1 - \frac{1}{|\Sigma|}\right)\cdot \frac{1}{2}\cdot \Pr\left(X_{\mathcal{Y}'} \geqslant 1 \,\middle|\, |\mathcal{Y}'| \geqslant |\Sigma|/16 \wedge \mathcal{C}_1\right) \\ &\geqslant \left(1 - \frac{1}{|\Sigma|}\right)\cdot \frac{1}{2}\cdot (1 - e^{-0.08}) .\end{aligned}$$

□

References

[1] Aamand, A., Das, D., Kipouridis, E., Knudsen, J. B. T., Rasmussen, P. M. R., and Thorup, M. No repetition: Fast and reliable sampling with highly concentrated hashing. *Proc. VLDB Endow. 15*, 13 (2022), 3989–4001.

[2] Aamand, A., Knudsen, J. B. T., Knudsen, M. B. T., Rasmussen, P. M. R., and Thorup, M. Fast hashing with strong concentration bounds. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing* (2020), pp. 1265–1278.

[3] Aamand, A., Knudsen, J. B. T., and Thorup, M. Load balancing with dynamic set of balls and bins. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021* (2021), S. Khuller and V. V. Williams, Eds., ACM, pp. 1262–1275.

[4] Amble, O., and Knuth, D. E. Ordered hash tables. *The Computer Journal 17*, 2 (1974), 135–142.

[5] Arbitman, Y., Naor, M., and Segev, G. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA* (2010), IEEE Computer Society, pp. 787–796.

[6] Bender, M. A., Kuszmaul, B. C., and Kuszmaul, W. Linear probing revisited: Tombstones mark the demise of primary clustering. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022* (2021), IEEE, pp. 1171–1182.

[7] Black, J. R., Martel, C. U., and Qi, H. Graph and hashing algorithms for modern architectures: Design and performance. In *Proc. 2nd International Workshop on Algorithm Engineering (WAE)* (1998), pp. 37–48.

[8] Celis, P., Larson, P.-A., and Munro, J. I. Robin hood hashing. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)* (1985), IEEE, pp. 281–288.

[9] Christiani, T., Pagh, R., and Thorup, M. From independence to expansion and back again. To appear, 2015.

[10] Dahlgaard, S., Knudsen, M. B. T., Rotenberg, E., and Thorup, M. Hashing for statistics over k-partitions. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science* (2015), IEEE, pp. 1292–1310.

[11] Dahlgaard, S., Knudsen, M. B. T., and Thorup, M. Practical hash functions for similarity estimation and dimensionality reduction. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA* (2017), I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., pp. 6615–6625.

[12] Dietzfelbinger, M., and auf der Heide, F. M. A new universal class of hash functions and dynamic hashing in real time. In *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings* (1990), M. Paterson, Ed., vol. 443 of *Lecture Notes in Computer Science*, Springer, pp. 6–19.

[13] Dietzfelbinger, M., and Rink, M. Applications of a splitting trick. In *Proc. 36th International Colloquium on Automata, Languages and Programming (ICALP)* (2009), pp. 354–365.

[14] Dietzfelbinger, M., and Weidling, C. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science 380*, 1 (2007), 47–68. Automata, Languages and Programming.

[15] Dietzfelbinger, M., and Woelfel, P. Almost random graphs with simple hash functions. In *Proc. 25th ACM Symposium on Theory of Computing (STOC)* (2003), pp. 629–638.

[16] Dietzfelbinger, M., and Woelfel, P. Almost random graphs with simple hash functions. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2003), STOC '03, Association for Computing Machinery, p. 629–638.

[17] Dubhashi, D., and Ranjan, D. Balls and bins: A study in negative dependence. *Random Structures & Algorithms 13*, 5 (1998), 99–124.

[18] Flajolet, P., Éric Fusy, Gandouet, O., and Meunier, F. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *In Analysis of Algorithms (AOFA)* (2007).

[19] Fotakis, D., Pagh, R., Sanders, P., and Spirakis, P. G. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst. 38*, 2 (2005), 229–248.

[20] Greenberg, S., and Mohri, M. Tight lower bound on the probability of a binomial exceeding its expectation. *Statistics & Probability Letters 86* (2014), 91–98.

[21] Heileman, G. L., and Luo, W. How caching affects hashing. In *Proc. 7th Workshop on Algorithm Engineering and Experiments (ALENEX)* (2005), p. 141–154.

[22] Houen, J. B. T., and Thorup, M. Understanding the moments of tabulation hashing via chaoses. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France* (2022), M. Bojanczyk, E. Merelli, and D. P. Woodruff, Eds., vol. 229 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 74:1–74:19.

[23] Klassen, T. Q., and Woelfel, P. Independence of tabulation-based hash classes. In *Proc. 10th Latin American Theoretical Informatics (LATIN)* (2012), pp. 506–517.

[24] Knudsen, M. B. T. Linear hashing is awesome. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)* (2016), IEEE, pp. 345–352.

[25] Knuth, D. E. Notes on open addressing. Unpublished memorandum. See `http://citeseer.ist.psu.edu/knuth63notes.html`, 1963.

[26] Li, P., Owen, A. B., and Zhang, C.-H. One permutation hashing. In *Proc. 26thAdvances in Neural Information Processing Systems* (2012), pp. 3122–3130.

[27] Mitzenmacher, M., and Vadhan, S. P. Why simple hash functions work: exploiting the entropy in a data stream. In *Proc. 19th ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2008), pp. 746–755.

[28] Pagh, A., and Pagh, R. Uniform hashing in constant time and optimal space. *SIAM J. Comput. 38*, 1 (2008), 85–96.

[29] Pagh, A., Pagh, R., and Ružić, M. Linear probing with constant independence. *SIAM Journal on Computing 39*, 3 (2009), 1107–1120. See also STOC'07.

[30] Pătraşcu, M., and Thorup, M. On the k-independence required by linear probing and minwise independence. In *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP)* (2010), pp. 715–726.

[31] Pătraşcu, M., and Thorup, M. The power of simple tabulation-based hashing. *Journal of the ACM 59*, 3 (2012), Article 14. Announced at STOC'11.

[32] Pătraşcu, M., and Thorup, M. Twisted tabulation hashing. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2013), pp. 209–228.

[33] Siegel, A. On universal classes of extremely random constant-time hash functions. *SIAM Journal on Computing 33*, 3 (2004), 505–543. See also FOCS'89.

[34] Thorup, M. Timeouts with time-reversed linear probing. In *Proc. IEEE INFOCOM* (2011), pp. 166–170.

[35] Thorup, M. Simple tabulation, fast expanders, double tabulation, and high independence. In *FOCS* (2013), pp. 90–99.

[36] Thorup, M., and Zhang, Y. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing 41*, 2 (2012), 293–331. Announced at SODA'04 and ALENEX'10.

[37] Wegman, M. N., and Carter, L. New classes and applications of hash functions. *Journal of Computer and System Sciences 22*, 3 (1981), 265–279. See also FOCS'79.

[38] Zobrist, A. L. A new hashing method with application for game playing. Tech. Rep. 88, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1970.

Appendix F

NeurIPS: Multi-Swap k-Means++

Multi-Swap k-Means++

Lorenzo Beretta[*]
University of Copenhagen
lorenzo2beretta@gmail.com

Vincent Cohen-Addad
Google Research
cohenaddad@google.com

Silvio Lattanzi
Google Research
silviol@google.com

Nikos Parotsidis
Google Research
nikosp@google.com

Abstract

The k-means++ algorithm of Arthur and Vassilvitskii (SODA 2007) is often the practitioners' choice algorithm for optimizing the popular k-means clustering objective and is known to give an $O(\log k)$-approximation in expectation. To obtain higher quality solutions, Lattanzi and Sohler (ICML 2019) proposed augmenting k-means++ with $O(k \log \log k)$ local search steps obtained through the k-means++ sampling distribution to yield a c-approximation to the k-means clustering problem, where c is a large absolute constant. Here we generalize and extend their local search algorithm by considering larger and more sophisticated local search neighborhoods hence allowing to swap multiple centers at the same time. Our algorithm achieves a $9 + \varepsilon$ approximation ratio, which is the best possible for local search. Importantly we show that our approach yields substantial practical improvements, we show significant quality improvements over the approach of Lattanzi and Sohler (ICML 2019) on several datasets.

1 Introduction

Clustering is a central problem in unsupervised learning. In clustering one is interested in grouping together "similar" object and separate "dissimilar" one. Thanks to its popularity many notions of clustering have been proposed overtime. In this paper, we focus on metric clustering and on one of the most studied problem in the area: the Euclidean k-means problem.

In the Euclidean k-means problem one is given in input a set of points P in $\mathbb{R}^d$. The goal of the problem is to find a set of k centers so that the sum of the square distances to the centers is minimized. More formally, we are interested in finding a set C of k points in $\mathbb{R}^d$ such that $\sum_{p \in P} \min_{c \in C} ||p - c||^2$, where with $||p - c||$ we denote the Euclidean distance between p and c.

The k-means problem has a long history, in statistics and operations research. For Euclidean k-means with running time polynomial in both n, k and d, a 5.912-approximation was recently shown in Cohen-Addad et al. [2022a], improving upon Kanungo et al. [2004], Ahmadian et al. [2019], Grandoni et al. [2022] by leveraging the properties of the Euclidean metric. In terms of lower bounds, the first to show that the high-dimensional k-means problems were APX-hard were Guruswami and Indyk [2003], and later Awasthi et al. [2015] showed that the APX-hardness holds even if the centers can be placed arbitrarily in $\mathbb{R}^d$. The inapproximability bound was later slightly improved by Lee et al. [2017] until the recent best known bounds of Cohen-Addad and Karthik C. S. [2019], Cohen-Addad et al. [2022d] that showed that it is NP-hard to achieve a better than 1.06-approximation and hard to approximate it better than 1.36 assuming a stronger conjecture. From a more practical point of view, Arthur and Vassilvitskii [2009] showed that the widely-used popular heuristic of Lloyd Lloyd [1957]

[*]Authors are ordered in alphabetical order.

Preprint. Under review.

can lead to solutions with arbitrarily bad approximation guarantees, but can be improved by a simple seeding strategy, called k-means++, so as to guarantee that the output is within an $O(\log k)$ factor of the optimum Arthur and Vassilvitskii [2007].

Thanks to its simplicity k-means++ is widely adopted in practice. In an effort to improve its performances Lattanzi and Sohler [2019], Choo et al. [2020] combine k-means++ and local search to efficiently obtain a constant approximation algorithm with good practical performance. These two studies show that one can use the k-means++ distribution in combination with a local search algorithm to get the best of both worlds: a practical algorithm with constant approximation guarantees.

However, the constant obtained in Lattanzi and Sohler [2019], Choo et al. [2020] is very large (several thousands in theory) and the question as whether one could obtain a practical algorithm that would efficiently match the $9 + \varepsilon$-approximation obtained by the $n^{O(d/\epsilon)}$ algorithm of Kanungo et al. [2004] has remained open. Bridging the gap between the theoretical approach of Kanungo et al. [2004] and k-means++ has thus been a long standing goal.

Our Contributions. We make significant progress on the above line of work.

- We adapt techniques from the analysis of Kanungo et al. [2004] to obtain a tighter analysis of the algorithm in Lattanzi and Sohler [2019]. In particular in Corollary 4, we show that their algorithm achieves an approximation of ratio of ≈ 26.64.
- We extend this approach to multi-swaps, where we allow swapping more than one center at each iteration of local search, improving significantly the approximation to ≈ 10.48 in time $O(nd \cdot poly(k))$.
- Leveraging ideas from Cohen-Addad et al. [2021], we design a better local search swap that improves the approximation further to $9 + \varepsilon$ (see Theorem 12). This new algorithm matches the $9 + \varepsilon$-approximation achieved by the local search algorithm in Kanungo et al. [2004], but it is significantly more efficient. Notice that 9 is the best approximation achievable through local search algorithms, as proved in Kanungo et al. [2004].
- We provide experiments where we compare against k-means++ and Lattanzi and Sohler [2019]. We study a variant of our algorithm that performs very competitively with our theoretically sound algorithm. The variant is very efficient and still outperforms previous work in terms of solution quality, even after the standard postprocessing using Lloyd.

Additional Related Work. We start by reviewing the approach of Kanungo et al. [2004] and a possible adaptation to our setting. The bound of $9 + \varepsilon$ on the approximation guarantee shown by Kanungo et al. [2004] is for the following algorithm: Given a set S of k centers, if there is a set S^+ of at most $2/\varepsilon$ points in $\mathbb{R}^d$ together with a set S^- of $|S^+|$ points in S such that $S \setminus S^- \cup S^+$ achieves a better k-means cost than S, then set $S := S \setminus S^- \cup S^+$ and repeat until convergence. The main drawback of the algorithm is that it asks whether there exists a set S^+ of points in $\mathbb{R}^d$ that could be swapped with elements of S to improve the cost. Identifying such a set, even of constant size, is already non-trivial. The best way of doing so is through the following path: First compute a coreset using the state-of-the-art coreset construction of Cohen-Addad et al. [2022b] and apply the dimensionality reduction of Becchetti et al. [2019], Makarychev et al. [2019], hence obtaining a set of $\tilde{O}(k/\varepsilon^4)$ points in dimension $O(\log k/\varepsilon^2)$. Then, compute grids using the discretization framework of Matousek [2000] to identify a set of $\varepsilon^{-O(d)} \sim k^{O(\varepsilon^{-2}\log(1/\varepsilon))}$ grid points that contains nearly-optimum centers. Now, run the local search algorithm where the sets S^+ are chosen from the grid points by brute-force enumeration over all possible subsets of grid points of size at most, say s. The running time of the whole algorithm with swaps of magnitude s, i.e.: $|S^+| \leq s$, hence becomes $k^{O(s\cdot\varepsilon^{-2}\log(1/\varepsilon))}$ for an approximation of $(1 + \varepsilon)(9 + 2/s)$, meaning a dependency in k of $k^{O(\varepsilon^{-3}\log(1/\varepsilon))}$ to achieve a $9 + \varepsilon$-approximation. Our results improves upon this approach in two ways: (1) it improves over the above theoretical bound and (2) does so through an efficient and implementable, i.e.: practical, algorithm.

Recently, Grunau et al. [2023] looked at how much applying a greedy rule on top of the k-means++ heuristic improves its performance. The heuristic is that at each step, the algorithm samples ℓ centers and only keeps the one that gives the best improvement in cost. Interestingly the authors prove that from a theoretical standpoint this heuristic does not improve the quality of the output. Local search algorithms for k-median and k-means have also been studied by Gupta and Tangwongsan [2008] who drastically simplified the analysis of Arya et al. [2004]. Cohen-Addad and Schwiegelshohn [2017] demonstrated the power of local search for stable instances. Friggstad et al. [2019], Cohen-Addad et al. [2019] showed that local search yields a PTAS for Euclidean inputs of bounded dimension (and doubling metrics) and minor-free metrics. Cohen-Addad [2018] showed how to speed up the local search algorithm using kd-trees (i.e.: for low dimensional inputs).

For fixed k, there are several known approximation schemes, typically using small coresets Becchetti et al. [2019], Feldman and Langberg [2011], Kumar et al. [2010]. The state-of-the-art approaches are due to Bhattacharya et al. [2020], Jaiswal et al. [2014]. The best known coreset construction remains Cohen-Addad et al. [2022c,b].

If the constraint on the number of output centers is relaxed, then we talk about bicriteria approximations and k-means has been largely studied Bandyapadhyay and Varadarajan [2016], Charikar and Guha [2005], Cohen-Addad and Mathieu [2015], Korupolu et al. [2000], Makarychev et al. [2016].

2 Preliminaries

Notation. We denote with $P \subseteq \mathbb{R}^d$ the set of input points and let $n = |P|$. Given a point set $Q \subseteq P$ we use $\mu(Q)$ to denote the mean of points in Q. Given a point $p \in P$ and a set of centers A we denote with $A[p]$ the closest center in A to p (ties are broken arbitrarily). We denote with $\mathcal{C}$ the set of centers currently found by our algorithm and with $\mathcal{O}^*$ an optimal set of centers. Therefore, given $p \in P$, we denote with $\mathcal{C}[p]$ and $\mathcal{O}^*[p]$ its closest ALG-center and OPT-center respectively. We denote by $\texttt{cost}(Q, A)$ the cost of points in $Q \subseteq P$ w.r.t. the centers in A, namely

$$\texttt{cost}(Q, A) = \sum_{q \in Q} \min_{c \in A} ||q - c||^2 .$$

We use ALG and OPT as a shorthand for $\texttt{cost}(P, \mathcal{C})$ and $\texttt{cost}(P, \mathcal{O}^*)$ respectively. When we sample points proportionally to their current cost (namely, sample q with probability $\texttt{cost}(q, \mathcal{C}) / \texttt{cost}(P, \mathcal{C})$) we call this the D^2 distribution. When using $O_\varepsilon(\cdot)$ and $\Omega_\varepsilon(\cdot)$ we mean that ε is considered constant. We use $\widetilde{O}(f)$ to hide polylogarithmic factors in f. The following lemma is folklore.

Lemma 1. *Given a point set $Q \subseteq P$ and a point $p \in P$ we have*

$$\texttt{cost}(Q, p) = \texttt{cost}(Q, \mu(Q)) + |Q| \cdot ||p - \mu(Q)||^2 .$$

Let O_i^* be an optimal cluster, we define the *radius* of O_i^* as ρ_i such that $\rho_i^2 \cdot |O_i^*| = \texttt{cost}(O_i^*, o_i)$, where $o_i = \mu(O_i^*)$. We define the *δ-core* of the optimal cluster O_i^* as the set of points $p \in O_i^*$ that lie in a ball of radius $(1 + \delta)\rho_i$ centered in o_i. In symbols, $\texttt{core}(O_i^*) = P \cap B(o_i, (1 + \delta)\rho_i)$. Throughout the paper, δ is always a small constant fixed upfront, hence we omit it.

Lemma 2. *Let O_i^* be an optimal cluster and sample $q \in O_i^*$ according to the D^2-distribution restricted to O_i^*. If $\texttt{cost}(O_i^*, \mathcal{C}) > (2 + 3\delta) \cdot \texttt{cost}(O_i^*, o_i)$ then $\Pr[q \in \texttt{core}(O_i^*)] = \Omega_\delta(1)$.*

Proof. Define $\alpha := \texttt{cost}(O_i^*, \mathcal{C}) / \texttt{cost}(O_i^*, o_i) > 2 + 3\delta$. Thanks to Lemma 1, for each $c \in \mathcal{C}$ we have $||c - o_i||^2 \geq (\alpha - 1)\rho_i^2$. Therefore, for each $y \in \texttt{core}(O_i^*)$ and every $c \in \mathcal{C}$ we have

$$\texttt{cost}(y, c) = ||y - c||^2 \geq \left(\sqrt{\alpha - 1} - (1 + \delta)\right)^2 \cdot \rho_i^2 = \Omega_\delta(\alpha \rho_i^2).$$

Moreover, by a Markov's inequality argument we have $|O_i^* \setminus \texttt{core}(O_i^*)| \leq \frac{1}{1+\delta} \cdot |O_i^*|$ and thus $|\texttt{core}(O_i^*)| \geq \Omega_\delta(|O_i^*|)$. Combining everything we get

$$\texttt{cost}(\texttt{core}(O_i^*), \mathcal{C}) \geq |\texttt{core}(O_i^*)| \cdot \min_{\substack{c \in \mathcal{C} \\ y \in \texttt{core}(O_i^*)}} \texttt{cost}(y, c) = \Omega_\delta(|O_i^*|) \cdot \Omega_\delta(\alpha \rho_i^2)$$

and $|O_i^*| \cdot \alpha \rho_i^2 = \texttt{cost}(O_i^*, \mathcal{C})$, hence $\texttt{cost}(\texttt{core}(O_i^*), \mathcal{C}) = \Omega_\delta(\texttt{cost}(O_i^*, \mathcal{C}))$. □

3 Multi-Swap k-Means++

The single-swap local search (SSLS) k-means++ algorithm in Lattanzi and Sohler [2019] works as follows. First, k centers are sampled using k-means++ (namely, they are sampled one by one according to the D^2 distribution, updated for every new center). Then, $O(k \log \log k)$ steps of local search follow. In each local search step a point $q \in P$ is D^2-sampled, then let c be the center among the current centers $\mathcal{C}$ such that $\texttt{cost}(P, (\mathcal{C} \setminus \{c\}) \cup \{q\})$ is minimum. If $\texttt{cost}(P, (\mathcal{C} \setminus \{c\}) \cup \{q\}) < \texttt{cost}(P, \mathcal{C})$ then we swap c and q, or more formally we set $\mathcal{C} \leftarrow (\mathcal{C} \setminus \{c\}) \cup \{q\}$.

We extend the SSLS so that we allow to swap multiple centers simultaneously and call this algorithm multi-swap local search (MSLS) k-means++. Swapping multiple centers at the same time achieves a lower approximation ratio, in exchange for a higher time complexity. In this section, we present and analyse the p-swap local search (LS) algorithm for a generic number of p centers swapped at each step. For any constant $\delta > 0$, we obtain an approximation ratio ALG/OPT $= \eta^2 + \delta$ where

$$\eta^2 - (2 + 2/p)\eta - (4 + 2/p) = 0. \tag{1}$$

The Algorithm. First, we initialize our set of centers using k-means++. Then, we run $O(ndk^{p-1})$ local search steps, where a local search step works as follows. We D^2-sample a set $In = \{q_1 \dots q_p\}$ of points from P (without updating costs). Then, we iterate over all possible sets $Out = \{c_1 \dots c_p\}$ of p distinct elements in $\mathcal{C} \cup In$ and select the set Out such that performing the swap (In, Out) maximally improves the cost[2]. If this choice of Out improves the cost, then we perform the swap (In, Out), else we do not perform any swap for this step.

Theorem 3. *For any $\delta > 0$, the p-swap local search algorithm above runs in $\widetilde{O}(ndk^{2p})$ time and, with constant probability, finds an $(\eta^2 + \delta)$-approximation of k-means, where η satisfies Equation* (1).

Notice that the SSLS algorithm of Lattanzi and Sohler [2019] is exactly the p-swap LS algorithm above for $p = 1$.

Corollary 4. *The single-swap local search in Lattanzi and Sohler [2019], Choo et al. [2020] achieves an approximation ratio < 26.64.*

Corollary 5. *For $p = O(1)$ large enough, multi-swap local search achieves an approximation ratio < 10.48 in time $O(nd \cdot poly(k))$.*

3.1 Analysis of Multi-Swap k-means++

In this section we prove Theorem 3. Our main stepping stone is the following lemma.

Lemma 6. *Let ALG denote the cost at some point in the execution of MSLS. As long as ALG/OPT $> \eta^2 + \delta$, a local search step improves the cost by a factor $1 - \Omega(1/k)$ with probability $\Omega(1/k^{p-1})$.*

Proof of Theorem 3. First, we show that $O(k^p \log\log k)$ local steps suffice to obtain the desired approximation ratio, with constant probability. Notice that a local search step can only improve the cost function, so it is sufficient to show that the approximation ratio is achieved at some point in time. We initialize our centers using k-means++, which gives a $O(\log k)$-approximation in expectation. Thus, using Markov's inequality the approximation guarantee $O(\log k)$ holds with arbitrary high constant probability. We say that a local-search step is *successful* if it improves the cost by a factor of at least $1 - \Omega(1/k)$. Thanks to Lemma 6, we know that unless the algorithm has already achieved the desired approximation ratio then a local-search step is successful with probability $\Omega(1/k^{p-1})$. To go from $O(\log k)$ to $\eta^2 + \delta$ we need $O(k \log\log k)$ successful local search steps. Standard concentration bounds on the value of a Negative Binomial random variable show that, with high probability, the number of trial to obtain $O(k \log\log k)$ successful local-search steps is $O(k^p \log\log k)$. Therefore, after $O(k^p \log\log k)$ local-search steps we obtain an approximation ratio of $\eta^2 + \delta$.

To prove the running time bound it is sufficient to show that a local search step can be performed in time $\widetilde{O}(ndk^{p-1})$. This is possible if we maintain, for each point $x \in P$, a dynamic sorted dictionary[3] storing the pairs $(\texttt{cost}(x, c_i), c_i)$ for each $c_i \in \mathcal{C}$. Then we can combine the exhaustive search over all possible size-p subsets of $\mathcal{C} \cup In$ and the computation of the new cost function using time $O(ndk^{p-1} \log k)$. To do so, we iterate over all possible size-$(p-1)$ subsets Z of $\mathcal{C} \cup In$ and update all costs as if these centers were removed, then for each point $x \in P$ we compute how much its cost increases if we remove its closest center c_x in $(\mathcal{C} \cup In) \setminus Z$ and charge that amount to c_x. In the end, we consider $Out = Z \cup \{c\}$ where c is the cheapest-to-remove center found in this way. □

The rest of this section is devoted to proving Lemma 6. For convenience, we prove that Lemma 6 holds whenever ALG/OPT $> \eta^2 + O(\delta)$, which is wlog by rescaling δ. Recall that we now focus on

[2]If $In \cap Out \neq \emptyset$ then we are actually performing the swap $(In \setminus Out, Out \setminus In)$ of size $< p$.

[3]Also known as dynamic predecessor search data structure.

a given step of the algorithm, and when we say current cost, current centers and current clusters we refer to the state of these objects at the end of the last local-search step before the current one. Let $O_1^* \dots O_k^*$ be an optimal clustering of P and let $\mathcal{O}^* = \{o_i = \mu(O_i^*) \mid \text{ for } i = 1 \dots k\}$ be the set of optimal centers of these clusters. We denote with $C_1 \dots C_k$ the current set of clusters at that stage of the local search and with $\mathcal{C} = \{c_1 \dots c_k\}$ the set of their respective current centers.

We say that c_i *captures* o_j if c_i is the closest current center to o_j, namely $c_i = \mathcal{C}[o_j]$. We say that c_i is *busy* if it captures more than p optimal centers, and we say it is *lonely* if it captures no optimal center. Let $\widetilde{\mathcal{O}} = \{o_i \mid \mathtt{cost}(O_i^*, \mathcal{C}) > \delta \cdot \mathrm{ALG}/k\}$ and $\widetilde{\mathcal{C}} = \mathcal{C} \setminus \{\mathcal{C}[o_i] \mid o_i \in \mathcal{O}^* \setminus \widetilde{\mathcal{O}}\}$. For ease of notation, we simply assume that $\widetilde{\mathcal{O}} = \{o_1 \dots o_h\}$ and $\widetilde{\mathcal{C}} = \{c_1 \dots c_{h'}\}$. Notice that $h' > h$.

Weighted ideal multi-swaps. Given $In \subseteq P$ and $Out \subseteq \widetilde{\mathcal{C}}$ of the same size we say that the swap (In, Out) is an *ideal* swap if $In \subseteq \widetilde{\mathcal{O}}$. We now build a set of *weighted* ideal multi-swaps $\mathcal{S}$. First, suppose wlog that $\{c_1 \dots c_t\}$ is the set of current centers in $\widetilde{\mathcal{C}}$ that are neither lonely nor busy. Let $\mathcal{L}$ be the set of lonely centers in $\widetilde{\mathcal{C}}$. For each $i = 1 \dots t$, we do the following. Let In be the set of optimal centers in $\widetilde{\mathcal{O}}$ captured by c_i. Choose a set $\mathcal{L}_i$ of $|In| - 1$ centers from $\mathcal{L}$, set $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathcal{L}_i$ and define $Out = \mathcal{L}_i \cup \{c_i\}$. Assign weight 1 to (In, Out) and add it to $\mathcal{S}$. For each busy center $c_i \in \{c_{t+1} \dots c_{h'}\}$ let A be the set of optimal centers in $\widetilde{\mathcal{O}}$ captured by c_i, pick a set $\mathcal{L}_i$ of $|A| - 1$ lonely current centers from $\mathcal{L}$ (a counting argument shows that this is always possible). Set $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathcal{L}_i$. For each $o_j \in A$ and $c_\ell \in \mathcal{L}_i$ assign weight $1/(|A| - 1)$ to (o_j, c_ℓ) and add it to $\mathcal{S}$. Suppose we are left with ℓ centers $o'_1 \dots o'_\ell \in \widetilde{\mathcal{O}}$ such that $\mathcal{C}[o'_i] \notin \widetilde{\mathcal{C}}$. Apparently, we have not included any o'_i in any swap yet. However, since $|\widetilde{\mathcal{C}}| \geq |\widetilde{\mathcal{O}}|$, we are left with at least $\ell' \geq \ell$ lonely centers $c'_1 \dots c'_{\ell'} \in \widetilde{\mathcal{C}}$. For each $i = 1 \dots \ell$ we assign weight 1 to (o'_i, c'_i) and add it to $\mathcal{S}$.

Observation 7. *The process above generates a set of weighted ideal multi-swaps such that: (i) Every swap has size at most p; (ii) The combined weights of swaps involving an optimal center $o_i \in \widetilde{\mathcal{O}}$ is* 1*; (iii) The combined weights of swaps involving a current center c_i is at most* $1 + 1/p$.

Consider an ideal swap (In, Out). Let $O_{In}^* = \bigcup_{o_i \in In} O_i^*$ and $C_{Out} = \bigcup_{c_j \in Out} C_j$. Define the reassignment cost $\mathtt{Reassign}(In, Out)$ as the increase in cost of reassigning points in $C_{Out} \setminus O_{In}^*$ to centers in $\mathcal{C} \setminus Out$. Namely,

$$\mathtt{Reassign}(In, Out) = \mathtt{cost}(C_{Out} \setminus O_{In}^*, \mathcal{C} \setminus Out) - \mathtt{cost}(C_{Out} \setminus O_{In}^*, \mathcal{C}).$$

We take the increase in cost of the following reassignment as an upper bound to the reassignment cost. For each $p \in C_{Out} \setminus O_{In}^*$ we consider its closest optimal center $\mathcal{O}^*[p]$ and reassign p to the current center that is closest to $\mathcal{O}^*[p]$, namely $\mathcal{C}[\mathcal{O}^*[p]]$. In formulas, we have

$$\begin{aligned}\mathtt{Reassign}(In, Out) &\leq \sum_{p \in C_{Out} \setminus O_{In}^*} \mathtt{cost}(p, \mathcal{C}[\mathcal{O}^*[p]]) - \mathtt{cost}(p, \mathcal{C}[p]) \\ &\leq \sum_{p \in C_{Out}} \mathtt{cost}(p, \mathcal{C}[\mathcal{O}^*[p]]) - \mathtt{cost}(p, \mathcal{C}[p]).\end{aligned}$$

Indeed, by the way we defined our ideal swaps we have $\mathcal{C}[\mathcal{O}^*[p]] \notin Out$ for each $p \notin O_{In}^*$ and this reassignment is valid. Notice that the right hand side in the equation above does not depend on In.

Lemma 8. $\sum_{p \in P} \mathit{cost}(p, \mathcal{C}[\mathcal{O}^*[p]]) \leq 2\mathit{OPT} + \mathit{ALG} + 2\sqrt{\mathit{ALG}}\sqrt{\mathit{OPT}}.$

Proof. Deferred to the supplementary material. □

Lemma 9. *The combined weighted reassignment costs of all ideal multi-swaps in $\mathcal{S}$ is at most* $(2 + 2/p) \cdot (\mathit{OPT} + \sqrt{\mathit{ALG}}\sqrt{\mathit{OPT}})$.

Proof. Denote by $w(In, Out)$ the weight associated with the swap (In, Out).

$$\sum_{(In,Out)\in\mathcal{S}} w(In, Out) \cdot \texttt{Reassign}(In, Out) \leq$$

$$\sum_{(In,Out)\in\mathcal{S}} w(In, Out) \cdot \sum_{p\in C_{Out}} \texttt{cost}(p, \mathcal{C}[\mathcal{O}^*[p]]) - \texttt{cost}(p, \mathcal{C}[p]) \leq$$

$$(1 + 1/p) \cdot \sum_{c_j\in\mathcal{C}} \sum_{p\in C_j} \texttt{cost}(p, \mathcal{C}[\mathcal{O}^*[p]]) - \texttt{cost}(p, \mathcal{C}[p]) \leq$$

$$(1 + 1/p) \cdot \left(\sum_{p\in P} \texttt{cost}(p, \mathcal{C}[\mathcal{O}^*[p]]) - \text{ALG} \right).$$

The second inequality uses (iii) from Observation 7. Applying Lemma 8 completes the proof. □

Recall the notions of radius and core of an optimal cluster introduced in Section 2. We say that a swap (In, Out) is *strongly improving* if $\texttt{cost}(P, (\mathcal{C} \cup In) \setminus Out) \leq (1 - \delta/k) \cdot \texttt{cost}(P, \mathcal{C})$. Let $In = \{o_1 \ldots o_s\} \subseteq \widetilde{\mathcal{O}}$ and $Out = \{c_1 \ldots c_s\} \subseteq \widetilde{\mathcal{C}}$ we say that an ideal swap (In, Out) is *good* if for every $q_1 \in \texttt{core}(o_1) \ldots q_s \in \texttt{core}(o_s)$ the swap $(\mathcal{Q}, Out)$ is strongly improving, where $\mathcal{Q} = \{q_1 \ldots q_s\}$. We call an ideal swap *bad* otherwise. We say that an optimal center $o_i \in \widetilde{\mathcal{O}}$ is good if that's the case for at least one of the ideal swaps it belongs to, otherwise we say that it is bad. Notice that each optimal center in $\widetilde{\mathcal{O}}$ is assigned to a swap in $\mathcal{S}$, so it is either good or bad. Denote with G the union of cores of good optimal centers in $\widetilde{\mathcal{O}}$.

Lemma 10. *If an ideal swap (In, Out) is bad, then we have*

$$\texttt{cost}(O^*_{In}, \mathcal{C}) \leq (2 + \delta)\texttt{cost}(O^*_{In}, \mathcal{O}^*) + \texttt{Reassign}(In, Out) + \delta \text{ALG}/k. \tag{2}$$

Proof. Let $In = \{o_1 \ldots o_s\}$, $\mathcal{Q} = \{q_1 \ldots q_s\}$ such that $q_1 \in \texttt{core}(o_1) \ldots q_s \in \texttt{core}(o_s)$. Then, by Lemma 1 $\texttt{cost}(O^*_{In}, \mathcal{Q}) \leq (2 + \delta)\texttt{cost}(O^*_{In}, \mathcal{O}^*)$. Moreover, $\texttt{Reassign}(In, Out) = \texttt{cost}(P \setminus O^*_{In}, \mathcal{C} \setminus Out) - \texttt{cost}(P \setminus O^*_{In}, \mathcal{C})$ because points in $P \setminus C_{Out}$ are not affected by the swap. Therefore, $\texttt{cost}(P, (\mathcal{C} \cup \mathcal{Q}) \setminus Out) \leq (2 + \delta)\texttt{cost}(O^*_{In}, \mathcal{O}^*) + \texttt{Reassign}(In, Out) + \texttt{cost}(P \setminus O^*_{In}, \mathcal{C})$. Suppose by contradiction that Equation (2) does not hold, then

$$\texttt{cost}(P, \mathcal{C}) - \texttt{cost}(P, (\mathcal{C} \cup \mathcal{Q}) \setminus Out) =$$

$$\texttt{cost}(P \setminus O^*_{In}, \mathcal{C}) + \texttt{cost}(O^*_{In}, \mathcal{C}) - \texttt{cost}(P, (\mathcal{C} \cup \mathcal{Q}) \setminus Out) \geq \delta \text{ALG}/k.$$

Hence, $(\mathcal{Q}, Out)$ is strongly improving and this holds for any choice of $\mathcal{Q}$, contradiction. □

Lemma 11. *If ALG/OPT $> \eta^2 + \delta$ then $\texttt{cost}(G, \mathcal{C}) = \Omega_\delta(\texttt{cost}(P, \mathcal{C}))$. Thus, if we D^2-sample q we have $P[q \in G] = \Omega_\delta(1)$.*

Proof. First, we observe that the combined current cost of all optimal clusters in $\mathcal{O}^* \setminus \widetilde{\mathcal{O}}$ is at most $k \cdot \delta\text{ALG}/k = \delta\text{ALG}$. Now, we prove that the combined current cost of all O^*_i such that o_i is bad is $\leq (1 - 2\delta)\text{ALG}$. Suppose, by contradiction, that it is not the case, then we have:

$$(1 - 2\delta)\text{ALG} < \sum_{\text{Bad } o_i\in\widetilde{\mathcal{O}}} \texttt{cost}(O^*_i, \mathcal{C}) \leq \sum_{\text{Bad } (In,Out)\in\mathcal{S}} w(In, Out) \cdot \texttt{cost}(O^*_{In}, \mathcal{C}) \leq$$

$$\sum_{\text{Bad } (In,Out)} w(In, Out) \cdot ((2 + \delta)\texttt{cost}(O^*_{In}, \mathcal{O}^*) + \texttt{Reassign}(In, Out) + \delta\text{ALG}/k) \leq$$

$$(2 + \delta)\text{OPT} + (2 + 2/p)\text{OPT} + (2 + 2/p)\sqrt{\text{ALG}}\sqrt{\text{OPT}} + \delta\text{ALG}.$$

The second and last inequalities make use of Observation 7. The third inequality uses Lemma 10.

Setting $\eta^2 = \text{ALG/OPT}$ we obtain the inequality $\eta^2 - (2 + 2/p \pm O(\delta))\eta - (4 + 2/p \pm O(\delta)) \leq 0$. Hence, we obtain a contradiction in the previous argument as long as $\eta^2 - (2 + 2/p \pm O(\delta))\eta - (4 + 2/p \pm O(\delta)) > 0$. A contradiction there implies that at least an δ-fraction of the current cost is due to points in $\bigcup_{\text{Good } o_i\in\widetilde{\mathcal{O}}} O^*_i$. We combine this with Lemma 2 and conclude that the total current cost of $G = \bigcup_{\text{Good } o_i\in\widetilde{\mathcal{O}}} \texttt{core}(O^*_i)$ is $\Omega_\delta(\texttt{cost}(P, \mathcal{C}))$. □

Finally, we prove Lemma 6. Whenever $q_1 \in G$ we have that $q_1 \in \texttt{core}(o_1)$ for some good o_1. Then, for some $s \le p$ we can complete o_1 with $o_2 \dots o_s$ such that $In = \{o_1 \dots o_s\}$ belongs to a good swap. Concretely, there exists $Out \subseteq \mathcal{C}$ such that (In, Out) is a good swap. Since $In \subset \widetilde{\mathcal{O}}$ we have $\texttt{cost}(O_i^*, \mathcal{C}) > \delta\text{OPT}/k$ for all $o_i \in In$, which combined with Lemma 2 gives that for $i = 2 \dots s$ $P[q_i \in \texttt{core}(o_i)] \ge \Omega_\delta(1/k)$. Hence, we have $P[q_i \in \texttt{core}(o_i)$ for $i = 1 \dots s] \ge \Omega_{\delta,p}(1/k^{p-1})$. Whenever we sample $q_1 \dots q_s$ from $\texttt{core}(o_1) \dots \texttt{core}(o_s)$, we have that $(\mathcal{Q}, Out)$ is strongly improving. Notice, however, that $(\mathcal{Q}, Out)$ is a s-swap and we may have $s < p$. Nevertheless, whenever we sample $q_1 \dots q_s$ followed by any sequence $q_{s+1} \dots q_p$ it is enough to choose $Out' = Out \cup \{q_{s+1} \dots q_p\}$ to obtain that $(\{q_1 \dots q_p\}, Out')$ is an improving p-swap.

4 A Faster $(9 + \varepsilon)$-Approximation Local Search Algorithm

The MSLS algorithm from Section 3 achieves an approximation ratio of $\eta^2 + \varepsilon$, where $\eta^2 - (2 + 2/p)\eta - (4 + 2/p) = 0$ and $\varepsilon > 0$ is an arbitrary small constant. For large p we have $\eta \approx 10.48$. On the other hand, employing p simultaneous swaps, Kanungo et al. [2004] achieve an approximation factor of $\xi^2 + \varepsilon$ where $\xi^2 - (2 + 2/p)\xi - (3 + 2/p) = 0$. If we set $p \approx 1/\varepsilon$ this yields a $(9 + O(\varepsilon))$-approximation. In the same paper, they prove that 9-approximation is indeed the best possible for p-swap local search, if p is constant (see Theorem 3.1 in Kanungo et al. [2004]). They showed that 9 is the right locality gap for local search, but they matched it with a very slow algorithm. To achieve a $(9 + \varepsilon)$-approximation, they discretize the space reducing to $O(n\varepsilon^{-d})$ candidate centers and perform an exhaustive search over all size-$(1/\varepsilon)$ subsets of candidates at every step. As we saw in the related work section, it is possible to combine techniques from coreset and dimensionality reduction to reduce the number of points to $n' = k \cdot poly(\varepsilon^{-1})$ and the number of dimensions to $d' = \log k \cdot \varepsilon^{-2}$. This reduces the complexity of Kanungo et al. [2004] to $k^{O(\varepsilon^{-3} \log \varepsilon^{-1})}$.

In this section, we leverage techniques from Cohen-Addad et al. [2021] to achieve a $(9 + \varepsilon)$-approximation faster [4]. In particular, we obtain the following.

Theorem 12. *Given a set of n points in $\mathbb{R}^d$ with aspect ratio Δ, there exists an algorithm that computes a $9 + \varepsilon$-approximation to k-means in time $ndk^{O(\varepsilon^{-2})} \log^{O(\varepsilon^{-1})}(\Delta) \cdot 2^{-poly(\varepsilon^{-1})}$.*

Notice that, besides being asymptotically slower, the pipeline obtained combining known techniques is highly impractical and thus it did not make for an experimental test-bed. Moreover, it is not obvious how to simplify such an ensemble of complex techniques to obtain a practical algorithm.

Limitations of MSLS. The barrier we need to overcome in order to match the bound in Kanungo et al. [2004] is that, while we only consider points in P as candidate centers, the discretization they employ considers also points in $\mathbb{R}^d \setminus P$. In the analysis of MSLS we show that we sample each point q_i from $\texttt{core}(O_i^*)$ or equivalently that $q_i \in B(o_i, (1 + \epsilon)\rho_i)$, where ρ_i is such that O_i^* would have the same cost w.r.t. o_i if all its points were moved on a sphere of radius ρ_i centered in o_i. This allows us to use a Markov's inequality kind of argument and conclude that there must be $\Omega_\epsilon(|O_i^*|)$ points in $O_i^* \cap B(o_i, (1 + \epsilon)\rho_i)$. However, we have no guarantee that there is any point at all in $O_i^* \cap B(o_i, (1 - \varepsilon)\rho_i)$. Indeed, all points in O_i^* might lie on $\partial B(o_i, \rho_i)$. The fact that potentially all our candidate centers q are at distance at least ρ_i from o_i yields (by Lemma 1) $\texttt{cost}(O_i^*, q) \ge 2\texttt{cost}(O_i^*, o_i)$, which causes the zero-degree term in $\xi^2 - (2+2/p)\xi - (3+2/p) = 0$ from Kanungo et al. [2004] to become a 4 in our analysis.

Improving MSLS by taking averages. First, we notice that, in order to achieve $(9 + \varepsilon)$-approximation we need to set $p = \Theta(1/\varepsilon)$. The main hurdle to achieve a $(9 + \varepsilon)$-approximation is that we need to replace the q_i in MSLS with a better approximation of o_i. We design a subroutine that computes, with constant probability, an ε-approximation $\hat{o}_i$ of o_i (namely, $\texttt{cost}(O_i^*, \hat{o}_i) \le (1 + \varepsilon)\texttt{cost}(O_i^*, o_i)$). The key idea is that, if sample uniformly $O(1/\varepsilon)$ points from O_i^* and define $\hat{o}_i$ to be the average of our samples then $\texttt{cost}(O_i^*, \hat{o}_i) \le (1 + \varepsilon)\texttt{cost}(O_i^*, o_i)$

Though, we do not know O_i^*, so sampling uniformly from it is non-trivial. To achieve that, for each q_i we identify a set N of *nice* candidate points in P such that a $poly(\varepsilon)/k$ fraction of them are from O_i^*. We sample $O(1/\varepsilon)$ points uniformly from N and thus with probability $(\varepsilon/k)^{O(1/\varepsilon)}$ we sample only points from O_i^*. Thus far, we sampled $O(1/\varepsilon)$ points uniformly from $N \cap O_i^*$. What about

[4]The complexity in Theorem 12 can be improved by applying the same preprocessing techniques using coresets and dimensionality reduction, similar to what can be used to speed up the approach of Kanungo et al. [2004]. Our algorithm hence becomes asymptotically faster.

the points in $O_i^* \setminus N$? We can define N so that all points in $O_i^* \setminus N$ are either very close to some of the $(q_j)_j$ or they are very far from q_i. The points that are very close to points $(q_j)_j$ are easy to treat. Indeed, we can approximately locate them and we just need to guess their mass, which is matters only when $\geq poly(\varepsilon)$ALG, and so we pay only a $\log^{O(1/\varepsilon)}(1/\varepsilon)$ multiplicative overhead to guess the mass close to q_j for $j = 1 \dots p = \Theta(1/\varepsilon)$. As for a point f that is very far from q_i (say, $||f - q_i|| \gg \rho_i$) we notice that, although f's contribution to $\texttt{cost}(O_i^*, o_i)$ may be large, we have $\texttt{cost}(f, o) \approx \texttt{cost}(f, o_i)$ for each $o \in B(q_i, \rho_i) \subseteq B(o_i, (2 + \varepsilon)\rho_i)$ assuming $q_i \in \texttt{core}(o_i)$.

5 Experiments

In this section, we show that our new algorithm using multi-swap local search can be employed to design an efficient seeding algorithm for Lloyd's which outperforms both the classical k-means++ seeding and the single-swap local search from Lattanzi and Sohler [2019].

Algorithms. The multi-swap local search algorithm that we analysed above performs very well in terms of solution quality. This empirically verifies the improved approximation factor of our algorithm, compared to the single-swap local search of Lattanzi and Sohler [2019].

Motivated by practical considerations, we heuristically adapt our algorithm to make it very competitive with SSLS in terms of running time and still remain very close, in terms of solution quality, to the theoretically superior algorithm that we analyzed. The adaptation of our algorithm replaces the phase where it selects the p centers to swap-out by performing an exhaustive search over $\binom{k+p}{p}$ subsets of centers. Instead, we use an efficient heuristic procedure for selecting the p centers to swap-out, by greedily selecting one by one the centers to swap-out. Specifically, we select the first center to be the cheapest one to remove (namely, the one that increases the cost by the least amount once the points in its cluster are reassigned to the remaining centers). Then, we update all costs and select the next center iteratively. After p repetitions we are done. We perform an experimental evaluation of the "greedy" variant of our algorithm compared to the theoretically-sound algorithm from Section 3 and show that employing the greedy heuristic does not measurably impact performance.

The four algorithms that we evaluate are the following: 1) **KM++:** The k-means++ from Arthur and Vassilvitskii [2007], 2) **SSLS:** The Single-swap local search method from Lattanzi and Sohler [2019], 3) **MSLS:** The multi-swap local search from Section 3, and 4) **MSLS-G:** The greedy variant of multi-swap local search as described above.

We use MSLS-G-$p = x$ and MSLS-$p = x$, to denote MSLS-G and MSLS with $p = x$, respectively. Notice that MSLS-G-$p = 1$ is exactly SSLS. Our experimental evaluation explores the effect of p-swap LS, for $p > 1$, in terms of solution cost and running time.

Datasets. We consider the three datasets used in Lattanzi and Sohler [2019] to evaluate the performance of SSLS: 1) KDD-PHY – $100,000$ points with 78 features representing a quantum physic task kdd [2004], 2) RNA - $488,565$ points with 8 features representing RNA input sequence pairs Uzilov et al. [2006], and 3) KDD-BIO – $145,751$ points with 74 features measuring the match between a protein and a native sequence kdd [2004]. We discuss the results for two or our datasets, namely KDD-BIO and RNA. We deffer the results on KDD-PHY to the appendix and note that the results are very similar to the results on RNA.

We performed a preprocessing step to clean-up the datasets. We observed that the standard deviation of some features was disproportionately high. This causes all costs being concentrated in few dimensions making the problem, in some sense, lower-dimensional. Thus, we apply min-max scaling to all datasets and observed that this causes all our features' standard deviations to be comparable.

Experimental setting. All our code is written in Python. The code will be made available upon publication of this work. We did not make use of parallelization techniques. To run our experiments, we used a personal computer with 8 cores, a 1.8 Ghz processor, and 15.9 GiB of main memory We run all experiments 5 times and report the mean and standard deviation in our plots. All our plots report the progression of the cost either w.r.t local search steps, or Lloyd's iterations. We run experiments on all our datasets for $k = 10, 25, 50$. The main body of the paper reports the results for $k = 25$, while the rest can be found in the appendix. We note that the conclusions of the experiments for $k = 10, 50$ are similar to those of $k = 25$.

Removing centers greedily. We first we compare MSLS-G with MSLS. To perform our experiment, we initialize $k = 25$ centers using k-means++ and then run 50 iterations of local search for both

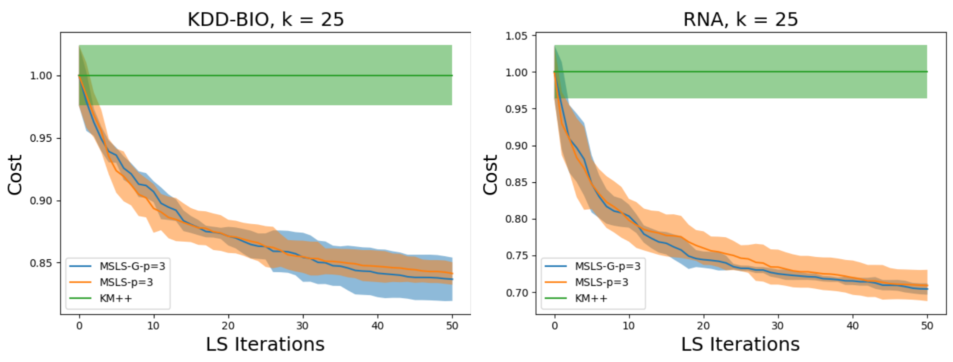


Figure 1: Comparison between MSLS and MSLS-G, for $p = 3$, for $k = 25$, on the datasets KDD-BIO and RNA. The y axis shows the solution cost divided by the means solution cost of KM++.

algorithms, for $p = 3$ swaps. Due to the higher running of the MSLS we perform this experiments on 1% uniform sample of each of our datasets. We find out that the performance of the two algorithms is comparable on all our instances, while they both perform roughly 15%-27% at convergence. Figure 1 shows the aggregate results, over 5 repetitions of our experiment.

It may happen that MSLS, which considers all possible swaps of size p at each LS iteration, performs worse than MSLS-G as a sub-optimal swap at intermediate iterations may still lead to a better local optimum by coincidence. Given that MSLS-G performs very comparably to MSLS, while it is much faster in practice, we use MSLS-G for the rest of our experiments where we compare to baselines. This allows us to consider higher values of p, without compromising much the running time.

Results: Evaluating the quality and performance of the algorithms. In our first experiment we run KM++ followed by 50 iterations of MSLS-G with $p = 1, 4, 7, 10$ and plot the relative cost w.r.t. KM++ at each iteration, for $k = 25$. The first row of Figure 2 plots the results. Our experiment shows that, after 50 iterations MSLS-G for $p = 4, 7, 10$ achieves improvements of roughly 10% compared to MSLS-G-$p = 1$ and of the order of $20\% - 30\%$ compared to KM++. We also report the time per iteration that each algorithm takes. For comparison, we report the running time of a single iteration of Lloyd's next to the dataset's name. It is important to notice that, although MSLS-G-$p = 1$ is faster, running more iterations MSLS-G-$p = 1$ is not sufficient to compete with MSLS-G when $p > 1$.

Results: Evaluating the quality after postprocessing using Lloyd. In our second experiment, we use KM++ and MSLS-G as a seeding algorithm for Lloyd's and measure how much of the performance improvement measured in the first experiment is retained after running Lloyd's. First, we initialize our centers using KM++ and the run 15 iterations of MSLS-G for $p = 1, 4, 7$. We measure the cost achieved by running 10 iterations of Lloyd's starting from the solutions found by MSLS-G as well as KM++. In Figure 2 (second row) we plot the results. Notice that, according to the running times from the first experiment, 15 iterations iterations of MSLS-G take less than 10 iterations of Lloyd's for $p = 4, 7$ (and also for $p = 10$, except on RNA). We observe that MSLS-G for $p > 1$ performs at least as good as SSLS from Lattanzi and Sohler [2019] and in some cases maintains non-trivial improvements.

Results: Evaluating the quality and performance of the algorithms against a fixed deadline. In this experiment we run KM++ followed by MSLS-G with $p = 1, 4, 7, 10$, for a set of fixed amounts of time. This setting allows the versions of MSLS-G with smaller swap size to perform more iterations compared to the versions of the algorithm with a larger swap size, as smaller swap size leads to lower running time per iteration. Let τ be the average time that Lloyd's algorithm requires to complete a simple iteration on a specific instance. We plot the cost of the solution produced by each algorithm after running $\lambda \times \tau$ for each $\lambda \in \{1, \cdots, 20\}$ in Figure 3. Our experiment shows that MSLS-G for $p = 4, 7, 10$ achieves improvements of more than 5% compared to MSLS-G-$p = 1$ even when compared against a fixed running time, and of the order of $20\% - 30\%$ compared to KM++.

Conclusion and Future Directions

We present a new algorithm for the k-means problem and we show that it outperforms theoretically and experimentally state-of-the-art practical algorithms with provable guarantees in terms of solution

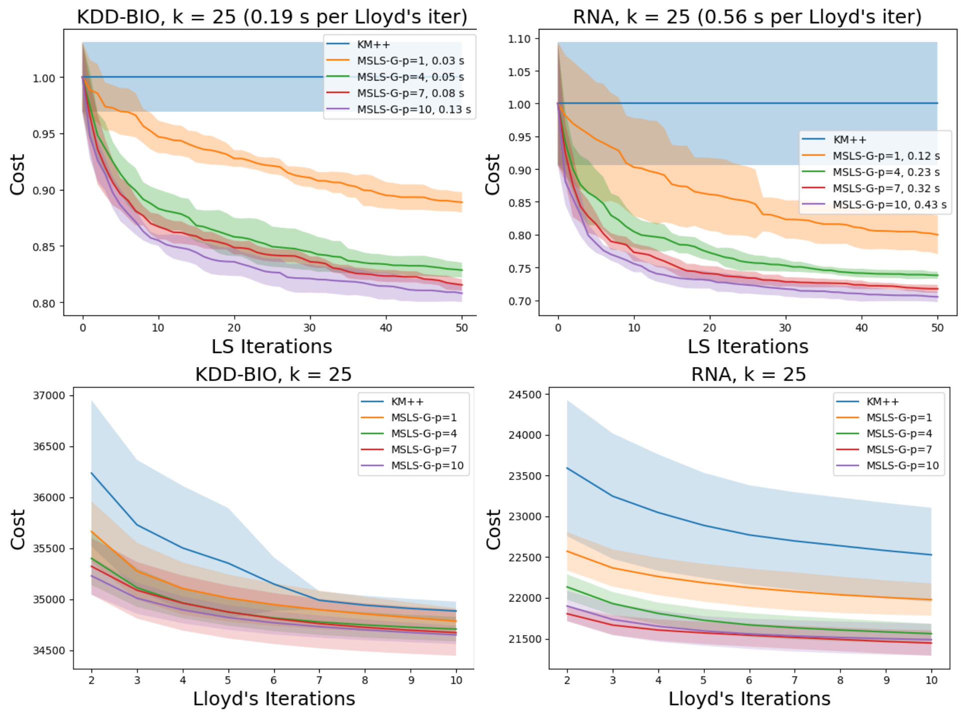


Figure 2: The first row compares the cost of MSLS-G, for $p \in \{1, 4, 7, 10\}$, divided by the mean cost of KM++ at each LS step, for $k = 25$. The legend reports also the running time of MSLS-G per LS step (in seconds). The second row compares the cost after each of the 10 iterations of Lloyd with seeding from MSLS-G, for $p \in \{1, 4, 7, 10\}$ and 15 local search steps and KM++, for $k = 25$.

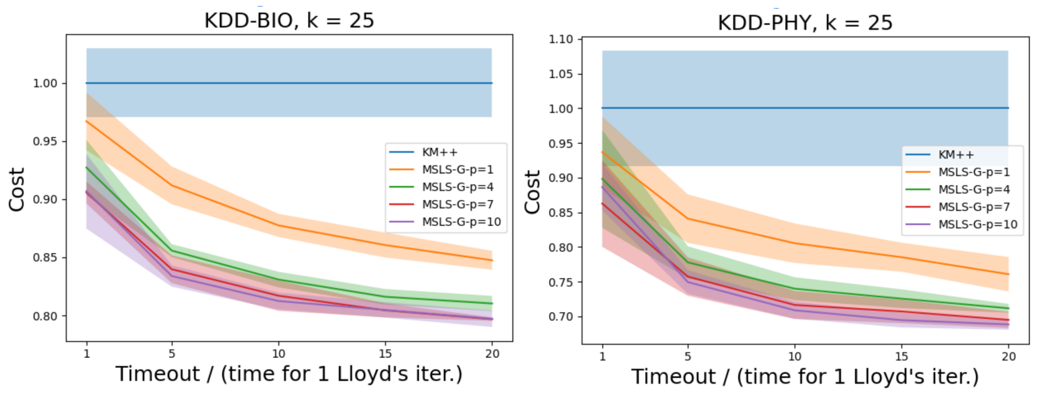


Figure 3: Comparison of the cost produced by MSLS-G, for $p \in \{1, 4, 7, 10\}$ and $k = 25$ on the datasets KDD-BIO and KDD-PHU, divided by the mean cost of KM++ after running for fixed amount of time in terms of multiplicative factors to the average time for an iteration of Lloyd's algorithm (i.e., for deadlines that are $1\times, \ldots, 20\times$ the average time of an iteration of Lloyd).

quality. A very interesting open question is to improve our local search procedure by avoiding the exhaustive search over all possible size-p subsets of centers to swap out, concretely an algorithm with running time $\tilde{O}(2^{poly(1/\varepsilon)} ndk)$.

References

Kdd cup. 2004. URL `http://osmot.cs.cornell.edu/kddcup/datasets.html`.

Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and Euclidean k-median by primal-dual algorithms. *SIAM Journal on Computing*, 49(4): FOCS17–97–FOCS17–156, 2019.

David Arthur and Sergei Vassilvitskii. K-means++ the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007.

David Arthur and Sergei Vassilvitskii. Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method. *SIAM J. Comput.*, 39(2):766–782, 2009. doi: 10.1137/070683921. URL `http://dx.doi.org/10.1137/070683921`.

Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004. doi: 10.1137/S0097539702416402. URL `https://doi.org/10.1137/S0097539702416402`.

Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of euclidean k-means. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, volume 34 of *LIPIcs*, pages 754–767. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi: 10.4230/LIPIcs.SOCG.2015.754. URL `https://doi.org/10.4230/LIPIcs.SOCG.2015.754`.

Sayan Bandyapadhyay and Kasturi Varadarajan. On variants of k-means clustering. In *32nd International Symposium on Computational Geometry (SoCG 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for k-means: beyond subspaces and the johnson-lindenstrauss lemma. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1039–1050, 2019. doi: 10.1145/3313276.3316318. URL `https://doi.org/10.1145/3313276.3316318`.

Anup Bhattacharya, Dishant Goyal, Ragesh Jaiswal, and Amit Kumar. On sampling based algorithms for k-means. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 13:1–13:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPIcs.FSTTCS.2020.13. URL `https://doi.org/10.4230/LIPIcs.FSTTCS.2020.13`.

Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM J. Comput.*, 34(4):803–824, 2005. doi: 10.1137/S0097539701398594. URL `https://doi.org/10.1137/S0097539701398594`.

Davin Choo, Christoph Grunau, Julian Portmann, and Vaclav Rozhon. k-means++: few more steps yield constant approximation. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1909–1917. PMLR, 2020. URL `https://proceedings.mlr.press/v119/choo20a.html`.

Vincent Cohen-Addad. A fast approximation scheme for low-dimensional k-means. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 430–440. SIAM, 2018. doi: 10.1137/1.9781611975031.29. URL `https://doi.org/10.1137/1.9781611975031.29`.

Vincent Cohen-Addad and Karthik C. S. Inapproximability of clustering in lp metrics. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 519–539. IEEE Computer Society, 2019. doi: 10.1109/FOCS.2019.00040. URL `https://doi.org/10.1109/FOCS.2019.00040`.

Vincent Cohen-Addad and Claire Mathieu. Effectiveness of local search for geometric optimization. In *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, pages 329–343, 2015. doi: 10.4230/LIPIcs.SOCG.2015.329. URL `http://dx.doi.org/10.4230/LIPIcs.SOCG.2015.329`.

Vincent Cohen-Addad and Chris Schwiegelshohn. On the local structure of stable clustering instances. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 49–60. IEEE Computer Society, 2017. doi: 10.1109/FOCS.2017.14. URL `https://doi.org/10.1109/FOCS.2017.14`.

Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. *SIAM J. Comput.*, 48(2): 644–667, 2019. doi: 10.1137/17M112717X. URL `https://doi.org/10.1137/17M112717X`.

Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. Improved coresets and sublinear algorithms for power means in euclidean spaces. *Advances in Neural Information Processing Systems*, 34:21085–21098, 2021.

Vincent Cohen-Addad, Hossein Esfandiari, Vahab S. Mirrokni, and Shyam Narayanan. Improved approximations for euclidean *k*-means and *k*-median, via nested quasi-independent sets. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1621–1628. ACM, 2022a. doi: 10.1145/3519935.3520011. URL `https://doi.org/10.1145/3519935.3520011`.

Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, and Chris Schwiegelshohn. Towards optimal lower bounds for k-median and k-means coresets. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1038–1051. ACM, 2022b. doi: 10.1145/3519935.3519946. URL `https://doi.org/10.1145/3519935.3519946`.

Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, Chris Schwiegelshohn, and Omar Ali Sheikh-Omar. Improved coresets for euclidean k-means. In *NeurIPS*, 2022c. URL `http://papers.nips.cc/paper_files/paper/2022/hash/120c9ab5c58ba0fa9dd3a22ace1de245-Abstract-Conference.html`.

Vincent Cohen-Addad, Euiwoong Lee, and Karthik C. S. Johnson coverage hypothesis: Inapproximability of k-means and k-median in ℓ_p metrics. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*. SIAM, 2022d.

D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *STOC*, pages 569–578, 2011.

Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for k-means in doubling metrics. *SIAM J. Comput.*, 48(2):452–480, 2019. doi: 10.1137/17M1127181. URL `https://doi.org/10.1137/17M1127181`.

Fabrizio Grandoni, Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Rakesh Venkat. A refined approximation for euclidean k-means. *Inf. Process. Lett.*, 176:106251, 2022. doi: 10.1016/j.ipl.2022.106251. URL `https://doi.org/10.1016/j.ipl.2022.106251`.

Christoph Grunau, Ahmet Alper Özüdogru, Václav Rozhon, and Jakub Tetek. A nearly tight analysis of greedy k-means++. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1012–1070. SIAM, 2023. doi: 10.1137/1.9781611977554.ch39. URL `https://doi.org/10.1137/1.9781611977554.ch39`.

Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *CoRR*, abs/0809.2554, 2008. URL `http://arxiv.org/abs/0809.2554`.

Venkatesan Guruswami and Piotr Indyk. Embeddings and non-approximability of geometric problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 537–538, 2003. URL `http://dl.acm.org/citation.cfm?id=644108.644198`.

Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 332–339, 1994.

Ragesh Jaiswal, Amit Kumar, and Sandeep Sen. A simple D 2-sampling based PTAS for k-means and other clustering problems. *Algorithmica*, 70(1):22–46, 2014. doi: 10.1007/s00453-013-9833-9. URL `https://doi.org/10.1007/s00453-013-9833-9`.

Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2):89–112, 2004. ISSN 0925-7721. doi: https://doi.org/10.1016/j.comgeo.2004.03.003. URL `https://www.sciencedirect.com/science/article/pii/S0925772104000215`. Special Issue on the 18th Annual Symposium on Computational Geometry - SoCG2002.

Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *J. Algorithms*, 37(1):146–188, 2000. doi: 10.1006/jagm.2000.1100. URL `http://dx.doi.org/10.1006/jagm.2000.1100`.

Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, 2010. doi: 10.1145/1667053.1667054. URL `https://doi.org/10.1145/1667053.1667054`.

Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3662–3671. PMLR, 2019. URL `https://proceedings.mlr.press/v97/lattanzi19a.html`.

Euiwoong Lee, Melanie Schmidt, and John Wright. Improved and simplified inapproximability for k-means. *Inf. Process. Lett.*, 120:40–43, 2017. doi: 10.1016/j.ipl.2016.11.009. URL `https://doi.org/10.1016/j.ipl.2016.11.009`.

SP Lloyd. Least square quantization in pcm. bell telephone laboratories paper. published in journal much later: Lloyd, sp: Least squares quantization in pcm. *IEEE Trans. Inform. Theor.(1957/1982)*, 18, 1957.

Konstantin Makarychev, Yury Makarychev, Maxim Sviridenko, and Justin Ward. A bi-criteria approximation algorithm for k-means. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPIcs*, pages 14:1–14:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi: 10.4230/LIPIcs.APPROX-RANDOM.2016.14. URL `https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2016.14`.

Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Performance of johnson-lindenstrauss transform for *k*-means and *k*-medians clustering. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1027–1038. ACM, 2019. doi: 10.1145/3313276.3316350. URL `https://doi.org/10.1145/3313276.3316350`.

Jirí Matousek. On approximate geometric k-clustering. *Discrete & Computational Geometry*, 24(1):61–84, 2000. doi: 10.1007/s004540010019. URL `http://dx.doi.org/10.1007/s004540010019`.

Andrew V Uzilov, Joshua M Keegan, and David H Mathews. Detection of non-coding rnas on the basis of predicted secondary structure formation free energy change. *BMC bioinformatics*, 7(1):1–30, 2006.

Supplementary Material

Proofs from Section 3

Lemma 8. $\sum_{p\in P} \mathtt{cost}(p, \mathcal{C}[\mathcal{O}^*[p]]) \le 2OPT + ALG + 2\sqrt{ALG}\sqrt{OPT}$.

Proof.

$$\sum_{p\in P} \mathtt{cost}(p, \mathcal{C}[\mathcal{O}^*[p]]) =$$
$$\sum_{o_i\in\mathcal{O}^*} \sum_{p\in O_i^*} \mathtt{cost}(p, \mathcal{C}[o_i]) =$$
$$\sum_{o_i\in\mathcal{O}^*} |O_i^*| \cdot \mathtt{cost}(o_i, \mathcal{C}[o_i]) + \mathtt{cost}(O_i^*, o_i) =$$
$$\text{OPT} + \sum_{p\in P} \mathtt{cost}(\mathcal{O}^*[p], \mathcal{C}[\mathcal{O}^*[p]]) \le$$
$$\text{OPT} + \sum_{p\in P} \mathtt{cost}(\mathcal{O}^*[p], \mathcal{C}[p]) \le$$
$$\text{OPT} + \sum_{p\in P} (||\mathcal{O}^*[p] - p|| + ||p - \mathcal{C}[p]||)^2 =$$
$$2\text{OPT} + \text{ALG} + 2\sum_{p\in P} ||\mathcal{O}^*[p], p|| \cdot ||p, \mathcal{C}[p]|| \le 2\text{OPT} + \text{ALG} + 2\sqrt{\text{ALG}}\sqrt{\text{OPT}}.$$

The second equality is due to Lemma 1 and the last inequality is due to Cauchy-Schwarz. □

Proofs from Section 4

In this section, we prove the following.

Theorem 12. *Given a set of n points in $\mathbb{R}^d$ with aspect ratio Δ, there exists an algorithm that computes a $9 + \varepsilon$-approximation to k-means in time $ndk^{O(\varepsilon^{-2})} \log^{O(\varepsilon^{-1})}(\Delta) \cdot 2^{-poly(\varepsilon^{-1})}$.*

We start with a key lemma showing that a sample of size $O(1/\varepsilon)$ is enough to approximate 1-mean.

Lemma 13 (Form Inaba et al. [1994]). *Given an instance $P \subseteq \mathbb{R}^d$, sample $m = 1/(\varepsilon\delta)$ points uniformly at random from P and denote the set of samples with S. Then $\mathtt{cost}(P, \mu(S)) \le (1 + \varepsilon)\mathtt{cost}(P, \mu(P))$ with probability at least $1 - \delta$.*

Proof. We want to prove that with probability $1 - \delta$ we have $||\mu(S) - \mu(P)||^2 \le \varepsilon\mathtt{cost}(P, \mu(P)) / |P|$. Then, applying Lemma 1 gives the desired result. First, we notice that $\mu(P)$ is an unbiased estimator of $\mu(P)$, namely $E[\mu(S)] = \mu(P)$. Then, we have

$$E\left[||\mu(S) - \mu(P)||^2\right] = \frac{1}{m}\sum_{i=1}^{|S|} E\left[||s_i - \mu(P)||^2\right] = \frac{\mathtt{cost}(P, \mu(P))}{m \cdot |P|}$$

where s_i are uniform independent samples from P. Applying Markov's inequality concludes the proof. □

The algorithm that verifies Theorem 12 is very similar to the MSLS algorithm from Section 3 and we use the same notation to describe it. The intuition is that in MSLS we sample $\mathcal{Q} = \{q_1 \dots q_p\}$ hoping that $q_i \in \mathtt{core}(o_i)$ for each i; here we refine q_i to a better approximation $\hat{o}_i$ of o_i and swap the points $(\hat{o}_i)_i$ rather than $(q_i)_i$. Our points $\hat{o}_i$ are generated taking the average of some sampled point, thus we possibly have $\hat{o}_i \notin P$ while, on the other hand, $q_i \in P$.

A $(9+\varepsilon)$-approximation MSLS algortihm. First, we initialize our set of centers using k-means++. Then, we run $ndk^{O(\varepsilon^{-2})} \cdot 2^{poly(\varepsilon^{-1})}$ local search steps, where a local search step works as follows. Set $p = \Theta(\varepsilon^{-1})$. We D^2-sample a set $\mathcal{Q} = \{q_1 \dots q_p\}$ of points from P (without updating costs). Then, we iterate over all possible sets $Out = \{c_1 \dots c_p\}$ of p distinct elements in $\mathcal{C} \cup \mathcal{Q}$. We define the set of *temporary* centers $\mathcal{T} = (\mathcal{C} \cup \mathcal{Q}) \setminus Out$ and run a subroutine APX-CENTERS$(\mathcal{T})$ which returns a list of $poly(\varepsilon^{-1}) \cdot \log^{O(\varepsilon^{-1})}(\Delta)$ size-s sets $\widehat{In} = \{\hat{o}_1 \dots \hat{o}_s\}$ (where $s = |\mathcal{Q} \setminus Out|$). We select the set $\widehat{In}$ in this list such that the swap $(\widehat{In}, Out \setminus \mathcal{Q})$ yields the maximum cost reduction. Then we select the set Out that maximizes the cost reduction obtained in this way. If $(\widehat{In}, Out \setminus \mathcal{Q})$ actually reduces the cost then we perform that swap.

A subroutine to approximate optimal centers. Here we describe the subroutine APX-CENTERS$(\mathcal{T})$. Let $\mathcal{Q} \setminus Out = \{q_1 \dots q_s\}$. Recall that $s \le p = O(\varepsilon^{-1})$. This subroutine outputs a list of $2^{poly(\varepsilon^{-1})} \cdot \log^{O(\varepsilon^{-1})}(\Delta)$ size-s sets $\widehat{In} = \{\hat{o}_1 \dots \hat{o}_s\}$. Here we describe how to find a list of $2^{poly(\varepsilon^{-1})} \cdot \log(\Delta)$ values for $\hat{o}_1$. The same will apply for $\hat{o}_2 \dots \hat{o}_s$ and taking the Cartesian product yields a list of $2^{poly(\varepsilon^{-1})} \cdot \log^{O(\varepsilon^{-1})}(\Delta)$ size-s sets. Assume wlog that the pairwise distances between points in P lie in $[1, \Delta]$. We iterate over all possible values of $\rho_1 \in \{1, (1+\varepsilon) \dots (1+\varepsilon)^{\lceil \log_{1+\varepsilon} \Delta \rceil}\}$. We partition P in three sets: the set of *far points* $F = \{x \in P \mid cost(x, q_1) > \rho_1^2/\varepsilon^3\}$, the set of *close points* $C = \{x \in P \setminus F \mid cost(x, \mathcal{T}) \le \varepsilon^3 \rho_1^2\}$ and the set of *nice points* $N = P \setminus (C \cup F)$. Then, we sample uniformly from N a set S of size $\Theta(\varepsilon^{-1})$. For each $(s+1)$-tuple of coefficients $\alpha_0, \alpha_1 \dots \alpha_s \in \left\{1, (1-\varepsilon), (1-\varepsilon)^2, \dots (1-\varepsilon)^{\lceil \log_{1-\varepsilon}(\varepsilon^7) \rceil}\right\} \cup \{0\}$ we output the candidate solution given by the convex combination

$$\hat{o}_1 = \hat{o}_1(\alpha_0 \dots \alpha_s) = \frac{\alpha_0 \mu(S) + \sum_{i=1}^{s} \alpha_i q_i}{\sum_{i=0}^{s} \alpha_i} \tag{3}$$

so, for each value of ρ_1, we output $2^{poly(\varepsilon^{-1})}$ values for $\hat{o}_1$. Hence, $2^{poly(\varepsilon^{-1})} \cdot \log(\Delta)$ values in total.

Analysis

The key insight in the analysis of the MSLS algorithm form Section 3 was that every q_i was a proxy for o_i because $q_i \in \texttt{core}(o_i)$, and thus q_i provided a good center for O_i^*. In the analysis of this improved version of MSLS we replace q_i with $\hat{o}_i$ which makes a better center for O_i^*. Formally, fixed Out, we say that a point $\hat{o}_i$ is a *perfect approximation* of o_i when $\texttt{cost}(O_i^*, (\mathcal{C} \cup \{\hat{o}_i\}) \setminus Out) \le (1+\varepsilon)\text{OPT}_i + \varepsilon \text{OPT}/k$. We define $\widetilde{\mathcal{O}}$ and $\widetilde{\mathcal{C}}$ as in Section 3, except that we replace δ with ε (which here is not assumed to be a constant). Likewise, we build the set $\mathcal{S}$ of ideal multi-swaps as in Section 3. Recall that we say that a multi-swap (In, Out) is *strongly improving* if $\texttt{cost}(P, (\mathcal{C} \cup In) \setminus Out) \le (1 - \varepsilon/k) \cdot \texttt{cost}(P, \mathcal{C})$. Let $In = \{o_1 \dots o_s\} \subseteq \widetilde{\mathcal{O}}$ and $Out = \{c_1 \dots c_s\} \subseteq \widetilde{\mathcal{C}}$, we overload the definition from Section 3 and say that the ideal multi-swap (In, Out) is *good* if for every $\widehat{In} = \{\hat{o}_1 \dots \hat{o}_s\}$ such that each $\hat{o}_i$ is a perfect approximation of o_i for each $i = 1 \dots s$ the swap $(\widehat{In}, Out)$ is strongly improving. We call an ideal swap *bad* otherwise. As in Section 3, we define the *core* of an optimal center; once again we replace δ with ϵ, which is no longer constant. The two following lemmas are our stepping stones towards Theorem 12.

Lemma 14. *If ALG/OPT $> 9 + O(\varepsilon)$ then, with probability $k^{-O(\varepsilon^{-1})} \cdot 2^{-poly(\varepsilon^{-1})}$, there exists $Out \subseteq \mathcal{C} \cup \mathcal{Q}$ such that:*

- *(i) If $\mathcal{Q} \setminus Out = \{q_1 \dots q_s\}$ then $q_1 \in \texttt{core}(o_1) \dots q_s \in \texttt{core}(o_s)$ for some $o_1 \dots o_s \in \mathcal{O}^*$*
- *(ii) If we define $In = \{o_1 \dots o_s\}$ then $(In, Out \setminus \mathcal{Q})$ is a good ideal swap.*

Lemma 15. *If (i) from Lemma 14 holds, then with probability $k^{-O(\varepsilon^{-2})} \cdot 2^{-poly(\varepsilon^{-1})}$, the list returned by* APX-CENTERS *contains $\widehat{In} = \{\hat{o}_1 \dots \hat{o}_s\}$ such that $\hat{o}_i$ is a perfect approximation of o_i for each $i = 1 \dots s$.*

Proof of Theorem 12. Here we prove that our improved MSLS algorithm achieves a $(9 + O(\varepsilon))$-approximation, which is equivalent to Theorem 12 up to rescaling ε. Combining Lemma 14 and

Lemma 15 we obtain that, as long as ALG/OPT $> 9 + O(\varepsilon)$, with probability at least $k^{-O(\varepsilon^{-2})} \cdot 2^{-poly(\varepsilon^{-1})}$, the list returned by APX-CENTERS contains $\widehat{In} = \{\hat{o}_1 \dots \hat{o}_s\}$ such that $(\widehat{In}, Out \setminus \mathcal{Q})$ is strongly improving. If this happens, we call such a local step *successful*. Now the proof goes exactly as the proof of Theorem 3. Indeed, We show that $k^{O(\varepsilon^{-2})} \cdot 2^{poly(\varepsilon^{-1})}$ local steps suffice to obtain $\Omega(k \log\log k/\varepsilon)$ successful local steps, and thus to obtain the desired approximation ratio, with constant probability.

To prove the running time bound it is sufficient to notice that a local search step can be performed in time $nd \log^{O(\varepsilon^{-1})}(\Delta) \cdot 2^{poly(\varepsilon^{-1})}$. □

In the rest of this section, we prove Lemma 14 and Lemma 15.

Observation 16. *If we assume $\delta = \varepsilon$ non-constant in Lemma 2, then performing the computations explicitly we obtain* $\Pr[q \in \mathtt{core}(O_i^*)] \geq poly(\varepsilon)$.

In order to prove Lemma 14, we first prove the two lemmas. Lemma 17 is the analogous of Lemma 10 and Lemma 18 is the analogous of Lemma 11. Overloading once again the definition from Section 3, we define G as the union of cores of good optimal centers in $\widetilde{\mathcal{O}}$, where an optimal center is defined to be good if at least one of the ideal multi-swaps in $\mathcal{S}$ it belongs to is good (exactly as in Section 3).

Lemma 17. *If an ideal swap (In, Out) is bad, then we have*

$$\mathtt{cost}(O_{In}^*, \mathcal{C}) \leq (1+\varepsilon)\mathtt{cost}(O_{In}^*, \mathcal{O}^*) + \mathtt{Reassign}(In, Out) + \varepsilon ALG/k. \tag{4}$$

Proof. Let $In = \{o_1 \dots o_s\}$, $\widehat{In} = \{\hat{o}_1 \dots \hat{o}_s\}$ such that $\hat{o}_i$ is a perfect approximation of o_i for each $i = 1 \dots s$. Recall that $O_{In}^* := \bigcup_{i=1}^s O_i^*$, then

$$\mathtt{cost}\Big(O_{In}^*, (\mathcal{C} \cup \widehat{In}) \setminus Out\Big) \leq \sum_{i=1}^s \mathtt{cost}(O_i^*, (\mathcal{C} \cup \{\hat{o}_i\}) \setminus Out) \leq (1+\varepsilon)\mathtt{cost}(O_{In}^*, \mathcal{O}^*)\,. \tag{5}$$

Moreover, $\mathtt{Reassign}(In, Out) = \mathtt{cost}(P \setminus O_{In}^*, \mathcal{C} \setminus Out) - \mathtt{cost}(P \setminus O_{In}^*, \mathcal{C})$ because points in $P \setminus C_{Out}$ are not affected by the swap. Therefore, $\mathtt{cost}\Big(P, (\mathcal{C} \cup \widehat{In}) \setminus Out\Big) \leq (1+\varepsilon)\mathtt{cost}(O_{In}^*, O^*) + \mathtt{Reassign}(In, Out) + \mathtt{cost}(P \setminus O_{In}^*, \mathcal{C})$. Suppose by contradiction that Equation (4) does not hold, then

$$\mathtt{cost}(P, \mathcal{C}) - \mathtt{cost}\Big(P, (\mathcal{C} \cup \widehat{In}) \setminus Out\Big) =$$
$$\mathtt{cost}(P \setminus O_{In}^*, \mathcal{C}) + \mathtt{cost}(O_{In}^*, \mathcal{C}) - \mathtt{cost}\Big(P, (\mathcal{C} \cup \widehat{In}) \setminus Out\Big) \geq \epsilon \text{ALG}/k.$$

Hence, $(\widehat{In}, Out)$ is strongly improving and this holds for any choice of $\widehat{In}$, contradiction. □

Lemma 18. *If ALG/OPT $> 9 + O(\varepsilon)$ then $\mathtt{cost}(G, \mathcal{C}) \geq \mathtt{cost}(P, \mathcal{C}) \cdot poly(\varepsilon)$. Thus, if we D^2-sample q we have $P[q \in G] \geq poly(\varepsilon)$.*

Proof. First, we observe that the combined current cost of all optimal clusters in $\mathcal{O}^* \setminus \widetilde{\mathcal{O}}$ is at most $k \cdot \varepsilon\text{ALG}/k = \varepsilon\text{ALG}$. Now, we prove that the combined current cost of all O_i^* such that o_i is bad is $\leq (1 - 2\varepsilon)\text{ALG}$. Suppose, by contradiction, that it is not the case, then we have:

$$(1-2\varepsilon)\text{ALG} < \sum_{\text{Bad } o_i \in \widetilde{\mathcal{O}}} \mathtt{cost}(O_i^*, \mathcal{C}) \leq \sum_{\text{Bad } (In, Out) \in \mathcal{S}} w(In, Out) \cdot \mathtt{cost}(O_{In}^*, \mathcal{C}) \leq$$
$$\sum_{\text{Bad } (In, Out)} w(In, Out) \cdot ((1+\varepsilon)\mathtt{cost}(O_{In}^*, \mathcal{O}^*) + \mathtt{Reassign}(In, Out) + \varepsilon\text{ALG}/k) \leq$$
$$(1+\varepsilon)\text{OPT} + (2 + 2/p)\text{OPT} + (2 + 2/p)\sqrt{\text{ALG}}\sqrt{\text{OPT}} + \varepsilon\text{ALG}.$$

The second and last inequalities make use of Observation 7. The third inequality uses Lemma 17.

Setting $\eta^2 = \text{ALG}/\text{OPT}$ we obtain the inequality $\eta^2 - (2 + 2/p \pm O(\varepsilon))\eta - (3 + 2/p \pm O(\varepsilon)) \leq 0$. Hence, we obtain a contradiction in the previous argument as long as $\eta^2 - (2 + 2/p \pm O(\varepsilon))\eta - (3 +$

$2/p \pm O(\varepsilon)) > 0$, which holds for $p = \Theta(\varepsilon^{-1})$ and $\eta^2 = 9 + O(\varepsilon)$. A contradiction there implies that at least an ε-fraction of the current cost is due to points in $\bigcup_{\text{Good } o_i \in \widetilde{\mathcal{O}}} O_i^*$. Thanks to Observation 16, we have $P_{q \sim \texttt{cost}(q,\mathcal{C})}[q \in \texttt{core}(O_i^*) \mid q \in O_i^*] \geq poly(\varepsilon)$. Therefore, we can conclude that the current cost of $G = \bigcup_{\text{Good } o_i \in \widetilde{\mathcal{O}}} \texttt{core}(O_i^*)$ is at least a $poly(\varepsilon)$-fraction of the total current cost. □

Proof of Lemma 14. Thanks to Lemma 18, we have that $P[q_1 \in G] \geq poly(\varepsilon)$. Whenever $q_1 \in G$ we have that $q_1 \in \texttt{core}(o_1)$ for some good o_1. Then, for some $s \leq p$ we can complete o_1 with $o_2 \dots o_s$ such that $In = \{o_1 \dots o_s\}$ belongs to a good swap. Concretely, there exists $Out \subseteq \mathcal{C}$ such that (In, Out) is a good swap. Since $In \subset \widetilde{\mathcal{O}}$ we have $\texttt{cost}(O_i^*, \mathcal{C}) > \varepsilon\text{OPT}/k$ for all $o_i \in In$, which combined with Observation 16 gives that, for each $i = 2 \dots s$, $P[q_i \in \texttt{core}(o_i)] \geq poly(\varepsilon)/k$. Hence, we have $P[q_i \in \texttt{core}(o_i) \text{ for } i = 1 \dots s] \geq 2^{-poly(\varepsilon^{-1})} k^{-O(\varepsilon^{-1})}$. Notice, however, that $(\widehat{In}, Out)$ is a s-swap and we may have $s < p$. Nevertheless, whenever we sample $q_1 \dots q_s$ followed by any sequence $q_{s+1} \dots q_p$ it is enough to choose $Out' = Out \cup \{q_{s+1} \dots q_p\}$ to obtain that $(\{q_1 \dots q_p\}, Out')$ is an improving p-swap. □

In order to prove Lemma 15 we first need a few technical lemmas.

Lemma 19 (Lemma 2 from Lattanzi and Sohler [2019]). *For each $x, y, z \in \mathbb{R}^d$ and $\varepsilon > 0$, $\texttt{cost}(x, y) \leq (1+\varepsilon)\texttt{cost}(x, z) + (1 + 1/\varepsilon)\texttt{cost}(z, y)$.*

Lemma 20. *Given $q \in \mathbb{R}^d$ and $Z \subseteq \mathbb{R}^d$ such that $\texttt{cost}(Z, q) \leq \varepsilon^2\Gamma$ then, for each $o \in \mathbb{R}^d$*

$$(1 - O(\varepsilon))\texttt{cost}(Z, o) - O(\varepsilon)\Gamma \leq |Z|\texttt{cost}(q, o) \leq (1 + O(\varepsilon))\texttt{cost}(Z, o) + O(\varepsilon)\Gamma$$

Proof. To obtain the first inequality, we apply Lemma 19 to bound $\texttt{cost}(z, o) \leq (1+\varepsilon)\texttt{cost}(z, o) + (1 + 1/\varepsilon)\texttt{cost}(z, q)$ for each $z \in Z$. To obtain the second inequality, we bound $\texttt{cost}(q, o) \leq (1+\varepsilon)\texttt{cost}(z, o) + (1 + 1/\varepsilon)\texttt{cost}(z, q)$ for each $z \in Z$. □

Lemma 21. *Let $X = \{x_1 \dots x_\ell\}$ be a weighted set of points in $\mathbb{R}^d$ such that x_i has weight w_i. Let μ be the weighted average of X. Let $\hat{\mu} = \hat{\mu}(\alpha_1 \dots \alpha_\ell)$ be the weighted average of X where x_i has weight α_i. If $w_i \leq \alpha_i \leq w_i/(1-\varepsilon)$ for each $i = 1 \dots \ell$, then if we interpret $\texttt{cost}(X, C)$ as $\sum_{x_i \in X} w_i \cdot \texttt{cost}(x_i, C)$ we have $\texttt{cost}(X, \hat{\mu}) \leq (1 + O(\varepsilon))\texttt{cost}(X, \mu)$.*

Proof. We note that μ minimizes the expression $\texttt{cost}(X, \mu)$. Moreover, $\texttt{cost}(X, z) \leq \sum_{i=1}^{\ell} \alpha_i \cdot \texttt{cost}(x_i, z) \leq \texttt{cost}(X, z)/(1-\varepsilon)$. Since $\hat{\mu}$ minimizes the expression $\sum_{i=1}^{\ell} \alpha_i \cdot \texttt{cost}(x_i, z)$ it must be $\texttt{cost}(X, \hat{\mu}) \leq \texttt{cost}(X, \mu)/(1-\varepsilon)$. □

Adopting the same proof strategy, we obtain the following.

Observation 22. *Thanks to Lemma 20, we can assume that the points in Z are concentrated in q for the purpose of computing a $(1 + O(\varepsilon))$-approximation to the 1-means problem on Z, whenever an additive error Γ is tolerable. Indeed, moving all points in Z to q introduces a $1 + O(\varepsilon)$ multiplicative error on $\texttt{cost}(Z, \cdot)$ and a $O(\varepsilon)\Gamma$ additive error.*

The next lemma shows that a point z that is far from a center o experiences a small variation of $\texttt{cost}(z, o)$ when the position of o is slightly perturbed.

Lemma 23. *Given $o, z \in \mathbb{R}^d$ such that $||o - z|| \geq r/\varepsilon$ we have that for every $o' \in B(o, r)$, $\texttt{cost}(z, o') = (1 \pm O(\varepsilon))\texttt{cost}(z, o)$.*

Proof. It is enough to prove it for all o' that lie on the line L passing through o and z, any other point in $o'' \in B(o, r)$ admits a point $o' \in B(o, r) \cap L$ with $||o' - z|| = ||o'' - z||$. It is enough to compute the derivative of $\texttt{cost}(z, \cdot)$ with respect to the direction of L and see that $\frac{\partial \texttt{cost}(z, \cdot)}{\partial L}|_{B(o,r)} = (1 \pm O(\varepsilon))r/\varepsilon$. Thus, $\texttt{cost}(z, o') = \texttt{cost}(z, o) \pm (1 \pm O(\varepsilon))r^2/\varepsilon = (1 \pm O(\varepsilon))\texttt{cost}(z, o)$. □

Proof of Lemma 15. Here we prove that for each $o_1 \dots o_s$ there exist coefficients $\alpha_0^{(i)} \dots \alpha_s^{(i)} \in \left\{1, (1-\varepsilon) \dots (1-\varepsilon)^{\lceil \log_{1-\varepsilon}(\varepsilon^7) \rceil}\right\} \cup \{0\}$ such that the convex combination $\hat{o}_i = \hat{o}_i(\alpha_0^{(i)} \dots \alpha_s^{(i)})$ is a perfect approximation of o_i, with probability $k^{-O(\varepsilon^{-2})} \cdot 2^{-poly(\varepsilon^{-1})}$. Wlog, we show this for o_1 only. Concretely, we want to show that, with probability $k^{-O(\varepsilon^{-1})} \cdot 2^{-poly(\varepsilon^{-1})}$, there exist coefficients $\alpha_0 \dots \alpha_s$ such that $\hat{o}_1 = \hat{o}_1(\alpha_0 \dots \alpha_s)$ satisfies $\texttt{cost}(O_1^*, (\mathcal{C} \cup \{\hat{o}_1\}) \setminus Out) \leq (1 + O(\varepsilon))\text{OPT}_1 + O(\varepsilon)\text{OPT}/k$. Taking the joint probability of these events for each $i = 1 \dots s$ we obtain the success probability $k^{-O(\varepsilon^{-2})} \cdot 2^{-poly(\varepsilon^{-1})}$. Note that we are supposed to prove that $\texttt{cost}(O_1^*, (\mathcal{C} \cup \{\hat{o}_1\}) \setminus Out) \leq (1+\varepsilon)\text{OPT}_1 + \varepsilon\text{OPT}/k$, however we prove a weaker version where ε is replaced by $O(\varepsilon)$, which is in fact equivalent up to rescaling ε.

Similarly to $\mathcal{C}[\cdot]$ and $\mathcal{O}^*[\cdot]$ define $\mathcal{T}[p]$ as the closest center to p in $\mathcal{T}$. Denote with C_1, F_1 and N_1 the intersections of O_1^* with C, F and N respectively. In what follows we define the values of $\alpha_0 \dots \alpha_s$ that define $\hat{o}_1 = \hat{o}_1(\alpha_0 \dots \alpha_s)$ and show an assignment of points in O_1^* to centers in $(\mathcal{C} \cup \{\hat{o}_1\}) \setminus Out$ with cost $(1 + O(\varepsilon))\text{OPT}_1 + O(\varepsilon)\text{OPT}/k$. Recall that we assume that $q_i \in \texttt{core}(o_i)$ for each $i = 1 \dots s$.

In what follows, we assign values to the coefficients $(\alpha_i)_i$. It is understood that if the final value we choose for α_i is v then we rather set α_i to the smallest power of $(1-\varepsilon)$ which is larger than v, if $v > \varepsilon^7$. Else, set α_i to 0. We will see in the end that this restrictions on the values of α_i do not impact our approximation.

In what follows, we will assign the points in O_1^* to $\mathcal{C} \setminus Out$, if this can be done inexpensively. If it cannot, then we will assign points to $\hat{o}_1$. In order to compute a good value for $\hat{o}_1$ we need an estimate of the average of points assigned to $\hat{o}_1$. For points in N_1, computing this average is doable (leveraging Lemma 13) while for points in $O_1^* \setminus N_1$ we show that either their contribution is negligible or we can collapse them so as to coincide with some $q_i \in \mathcal{Q}$ without affecting our approximation. The coefficients $(\alpha_i)_{i \geq 1}$ represent the fraction of points in O_i^* which is collapsed to q_i. α_0 represents the fraction of points in O_i^* which average we estimate as $\mu(S)$. Thus, Equation (3) defines $\hat{o}_i$ as the weighted average of points q_i, where the weights are the (approximate) fractions of points collapsed onto q_i, together with the the average $\mu(S)$ and its associated weight α_0.

Points in C_1. All points $p \in C_1$ such that $\mathcal{T}[p] \notin \mathcal{Q}$ can be assigned to $\mathcal{T}[p] \in \mathcal{C} \setminus Out$ incurring a total cost of at most $\varepsilon^6\text{OPT}_1$, by the definition of C_1. Given a point $p \in C_1$ with $\mathcal{T}[p] \in \mathcal{Q}$ we might have $\mathcal{T}[p] \notin \mathcal{C} \setminus Out$ and thus we cannot assign p to $\mathcal{T}[p]$. Denote with W the set of points p with $\mathcal{T}[p] \in \mathcal{Q}$. Our goal is now to approximate $\mu(W)$. In order to do that, we will move each $p \in W$ to coincide with $q_i = \mathcal{T}[p]$. We can partition W into $W_1 \dots W_s$ so that for each $z \in W_i$ $\mathcal{T}[z] = q_i$. If $p \in Z_i$ then we have $||p - q_i||^2 \leq \varepsilon^3 \rho_1^2$. Hence, thanks to Observation 22, we can consider points in W_i as if they were concentrated in q_i while losing at most an additive factor $O(\varepsilon)\text{OPT}_1$ and a multiplicative factor $(1+\varepsilon)$ on their cost. For $i = 1 \dots s$, set $\alpha_i \leftarrow |W_i|/|O_1^*|$. In this way, $\sum_{i=1}^{s} \alpha_i \cdot q_i / \sum_{i=1}^{s} \alpha_i$ is an approximates solution to 1-mean on W up to a multiplicative factor $(1+\varepsilon)$ and an additive factor $O(\varepsilon)\text{OPT}_1$.

Points in N_1. Consider the two cases: (i) $\texttt{cost}(N_1, \mathcal{T}) > \varepsilon^2\text{OPT}/k$; (ii) $\texttt{cost}(N_1, \mathcal{T}) \leq \varepsilon^2\text{OPT}/k$.

Case (i). We show that in this case $\mu(S)$ is a $(1+\varepsilon)$-approximation for 1-mean on N_1, with probability $k^{-O(\varepsilon^{-1})} \cdot 2^{-poly(\varepsilon^{-1})}$. First, notice that if we condition on $S \subseteq N_1$ then Lemma 13 gives that $\mu(S)$ is a $(1+\varepsilon)$-approximation for 1-mean on N_1 with constant probability. Thus, we are left to prove that $S \subseteq N_1$ with probability $k^{-O(\varepsilon^{-1})} \cdot 2^{-poly(\varepsilon^{-1})}$. We have that the $P_{p \sim \texttt{cost}(p, \mathcal{T})}[p \in N_1 \mid p \in N] \geq \varepsilon^2/k$, however the costs w.r.t. $\mathcal{T}$ of points in N varies of at most a factor $poly(\varepsilon^{-1})$, thus $P_{p \sim Unif}[p \in N_1 \mid p \in N] \geq poly(\varepsilon)/k$. The probability of $S \subseteq N_1$ is thus $(poly(\varepsilon)/k)^{|S|} = k^{-O(\varepsilon^{-1})} \cdot 2^{-poly(\varepsilon^{-1})}$. In this case, we set $\alpha_0 \leftarrow |N_1|/|O_1^*|$ because $\mu(S)$ approximates the mean of the entire set N_1.

Case (ii). Here we give up on estimating the mean of N_1 and set $\alpha_0 \leftarrow 0$. The point $x \in N_1$ such that $\mathcal{T}[x] \notin \mathcal{Q}$ can be assigned to $\mathcal{T}[x]$ incurring a combined cost of $\varepsilon^2\text{OPT}/k$. We partition the remaining points in N_1 into $Z_1 \cup \dots Z_s$ where each point x is placed in Z_i if $\mathcal{T}[x] = q_i$. Now, we collapse the points in Z_i so as to coincide with q_i and show that this does not worsen our approximation factor. In terms of coefficients $(\alpha_i)_i$, this translates into the updates $\alpha_i \leftarrow \alpha_i + |Z_i|/|O_i^*|$ for each $i = 1 \dots s$.

Indeed, using Observation 22 we can move all points in Z_i to q_i incurring an additive combined cost of $\varepsilon\text{OPT}/k$ and a multiplicative cost of $1 + O(\varepsilon)$.

Points in F_1. Points in F_1 are very far from q_1 and thus far from o_1, hence even if their contribution to $\texttt{cost}(O_1^*, o_1)$ might be large, we have $\texttt{cost}(F_1, o_1) = (1 \pm O(\varepsilon))\texttt{cost}(F_1, o')$ for all o' in a ball of radius ρ_1/ε centered in o_1, thanks to Lemma 23.

Let H be the set of points that have not been assigned to centers in $\mathcal{C} \setminus Out$. In particular, $H = W \cup N_1$ if points in N_1 satisfy case (i) and $H = W \cup Z_1 \dots Z_s$ if points in N_1 satisfy case (ii). We consider two cases.

If $||\mu(H) - q_1|| \leq \rho/\varepsilon$, then $||\mu(H) - o_1|| \leq \rho(1 + \varepsilon + 1/\varepsilon)$ because $q_1 \in \texttt{core}(o_1)$. Since for each $f \in F_1$ we have $||f - o_1|| \geq ||f - q_1|| - (1+\varepsilon)\rho \geq \Omega(\rho/\varepsilon^3)$ then $\texttt{cost}(f, o') = (1 \pm O(\varepsilon))\texttt{cost}(f, o_1)$ for each o' in a ball of radius $O(\rho/\varepsilon)$ centered in o_1, and so in particular for $o' = \mu(H)$. Thus in this case we can simply disregard all points in F_1 and computing $\hat{o}_1$ according to the $(\alpha_i)_i$ defined above yields a perfect approximation of o_i.

Else, if $||\mu(H) - q_1|| > \rho/\varepsilon$, a similar argument applies to show that $\texttt{cost}(H, o') = (1 \pm \varepsilon)\texttt{cost}(H, o)$ for each o' in ball of radius $O(\rho)$ centered in o_1. Indeed, we can rewrite $\texttt{cost}(H, o')$ as $|H| \cdot \texttt{cost}(\mu(H), o') + \texttt{cost}(\mu(H), H)$. If $||\mu(H) - q_1|| < \rho/\varepsilon$ the first term varies of at most a factor $(1+\varepsilon)$ and the second term is constant. Thus in this case $\hat{o}_1 = q_1$ is a perfect approximation of o_1 and we simply set $\alpha_1 = 1$ and $\alpha_j = 0$ for $j \neq 1$. In other words, here $\mu(N_1 \cup H)$ is too far from q_1 (and thus o_1) to significantlyt influence the position of $\hat{o}_1$ and the same holds for any point in F_1. This works, of course, because we assumed $q_1 \in \texttt{core}(o_1)$. □

Discussing the limitations on the coefficients values. The proof above would work smoothly if we were allowed to set α_i to exactly the values discussed above, representing the fractions of points from O_i^* captured by different q_is. However, to make the algorithm efficient we limit ourselves to values in $\left\{1, (1-\varepsilon) \dots (1-\varepsilon)^{\lceil \log_{1-\varepsilon}(\varepsilon^7) \rceil}\right\} \cup \{0\}$. Lemma 21 shows that as long as the values of $(\alpha_i)_i$ estimate the frequencies described above up to a factor $1 \pm O(\varepsilon)$ then the approximation error is within a multiplicative factor $1 \pm O(\varepsilon)$.

We are left to take care of the case in which α_i is set to a value $< \varepsilon^7$. We set α_i when dealing with points in $C_1 \cup N_1$ and for each $x \in C_1 \cup N_1$ we have, for each $o' \in B(q_1, (1+\varepsilon)\rho)$, $\texttt{cost}(x, o') \leq 2\texttt{cost}(q_1, o') + 2\texttt{cost}(x, q_1) = O(\rho_1 \varepsilon^{-6})$. Thus, if we simply set $\alpha_i \leftarrow 0$ whenever we have $\alpha_i < \varepsilon^7$ then the combined cost of points in O_1^* with respect to o' varies by $\varepsilon^7 |O_1^*| \cdot \rho_1 \varepsilon^{-6} = O(\varepsilon)\text{OPT}_1$. Effectively, ignoring these points does not significantly impact the cost. hence solving 1-mean ignoring these points finds a $(1 + O(\varepsilon))$-approximate solution to the original problem.

Additional Experimental Evaluation

In this section we report additional experiments which presentation did not fit in the main body. In particular, we run experiments on the dataset KDD-PHY and for $k = 10, 50$.

In Figure 4 we compare MSLS-G with MSLS. To perform our experiment, we initialize $k = 25$ centers using KM++ and then run 50 iterations of local search for both algorithms, for $p \in \{2, 3\}$ swaps. We repeat each experiment 5 times. For ease of comparison, we repeat the plot for the KDD-BIO and RNA datasets that we present in the main body of the paper. Due to the higher running of the MSLS we perform this experiments on 1% uniform sample of each of our datasets. We find out that the performance of the two algorithms is comparable on all our instances, while they both perform roughly 15%-27% better than k-means++ at convergence.

In Figure 5 we run KM++ followed by 50 iterations of MSLS-G with $p = 1, 4, 7, 10$ and $k = 10, 25, 50$ (expcluding the degenerate case $p = k = 10$) and plot the relative cost w.r.t. KM++ at each iteration. The results for $k = 25$ on KDD-BIO and RNA can be found in Figure 2. We repeat each experiment 5 times. Our experiment shows that, after 50 iterations MSLS-G for $p = 4, 7, 10$ achieves improvements of roughly $5 - 10\%$ compared to MSLS-G-$p = 1$ and of the order of $20\% - 40\%$ compared to KM++. These improvements are more prominent for $k = 25, 50$. We also report the time per iteration that each algorithm takes. For comparison, we report the running time of a single iteration of Lloyd's next to the dataset's name. Notice that the experiment on RNA for $k = 50$ is performed on a 10% uniform sample of the original dataset, due to the high running time.

In Figure 6, we use KM++ and MSLS-G as a seeding algorithm for Lloyd's and measure how much of the performance improvement measured is retained after running Lloyd's. First, we initialize

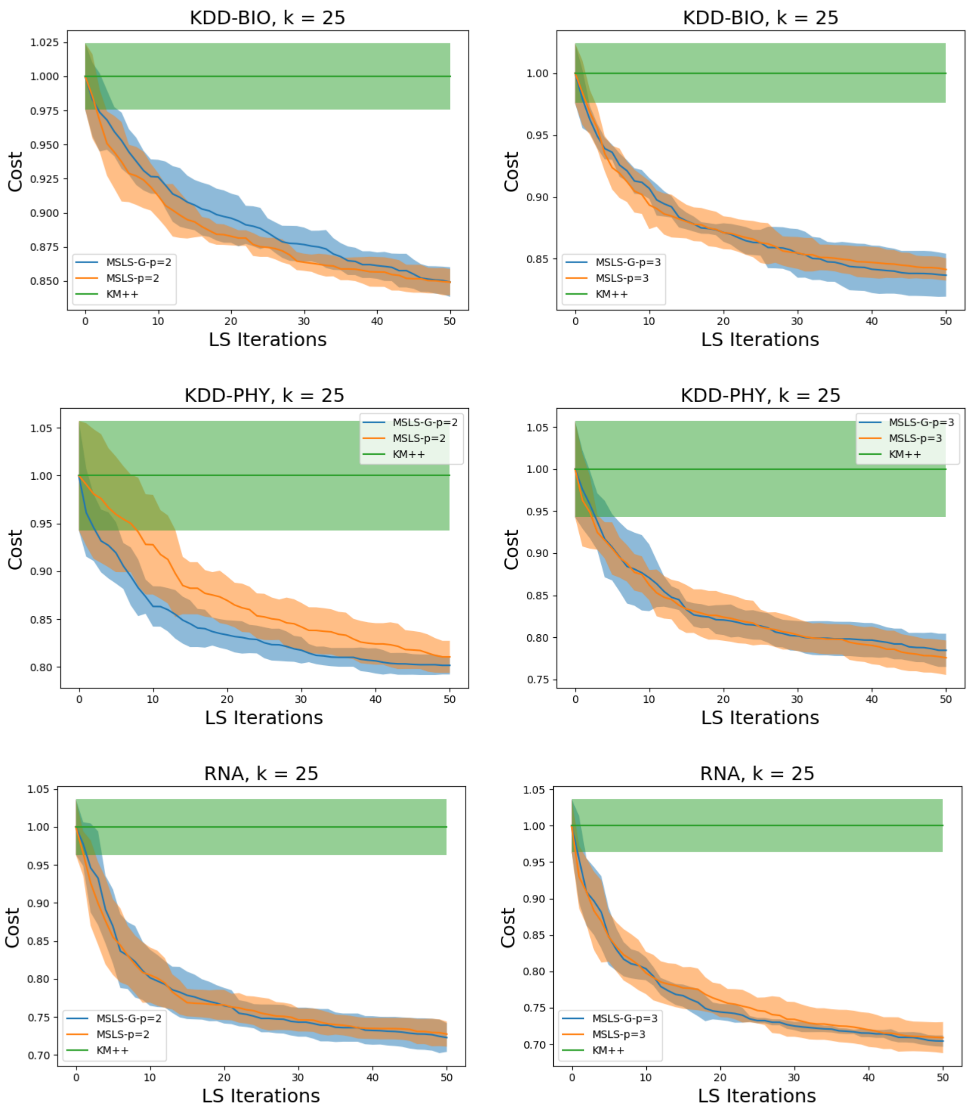


Figure 4: Comparison between MSLS and MSLS-G, for $p = 2$ (left column) and $p = 3$ (right column), for $k = 25$, on the datasets KDD-BIO (first row), KDD-PHY (second row) and RNA (third row). The y axis shows the mean solution cost, over the 5 repetitions of the experiment, divided by the means solution cost of KM++.

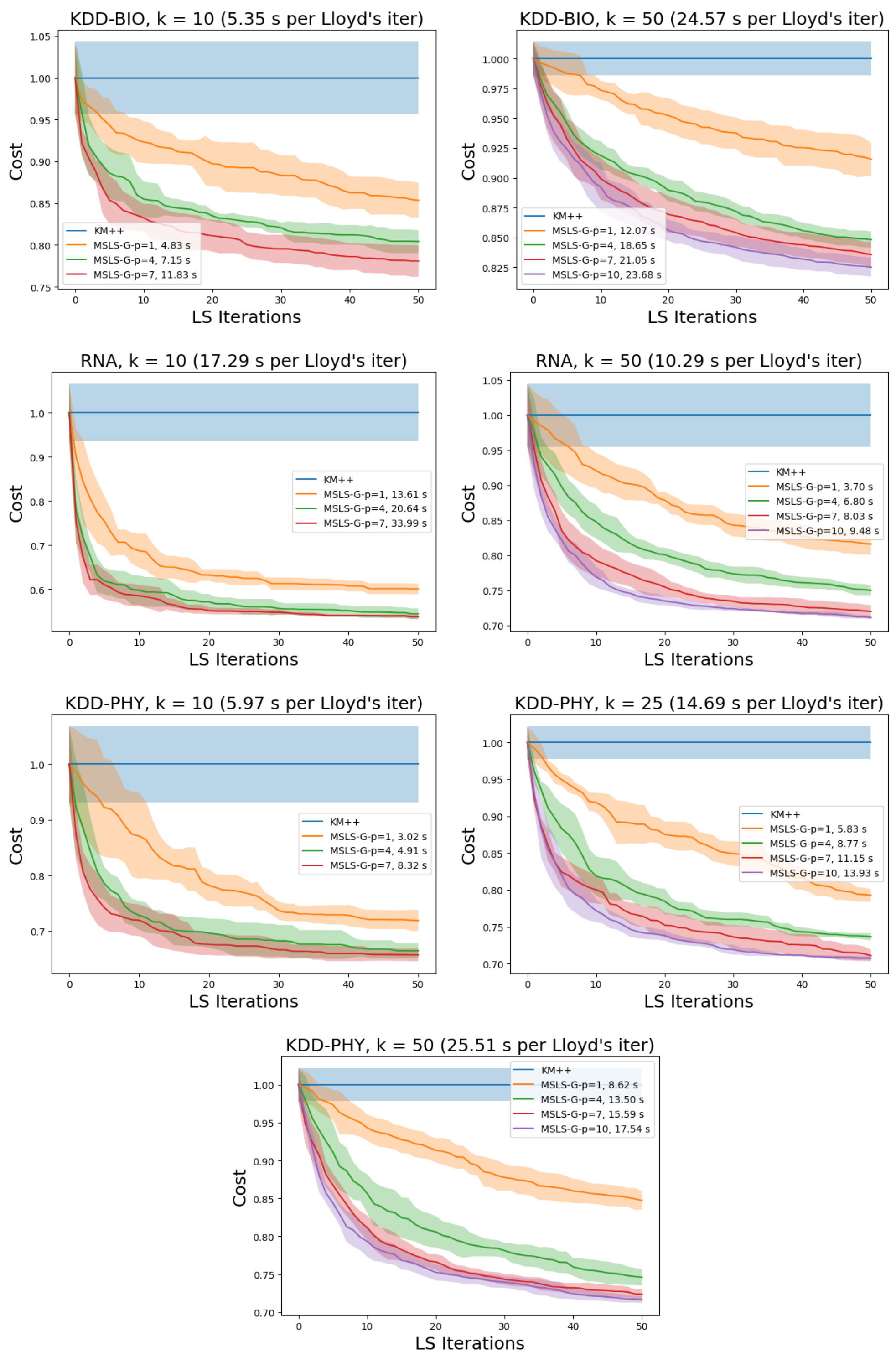


Figure 5: We compare the cost of MSLS-G, for $p \in \{1, 4, 7, 10\}$, divided by the mean cost of KM++ at each LS step, for $k \in \{10, 25, 50\}$, excluding the degenerate case $p = k = 10$. The legend reports also the running time of MSLS-G per LS step (in seconds). The experiments were run on all datasets: KDD-BIO, RNA and KDD-PHY, excluding the case of $k = 25$ for KDD-BIO and RNA which are reported in the main body of the paper.

our centers using KM++ and the run 15 iterations of MSLS-G for $p = 1, 4, 7$. We measure the cost achieved by running 10 iterations of Lloyd's starting from the solutions found by MSLS-G as well as KM++. We run experiments for $k = 10, 25, 50$ and we repeat each experiment 5 times. We observe that for $k = 25, 50$ MSLS-G for $p > 1$ performs at least as good as SSLS from Lattanzi and Sohler [2019] and in some cases maintains non-trivial improvements. These improvements are not noticeable for $k = 10$; however, given how Lloyd's behave for $k = 10$ we conjecture that $k = 10$ might be an "unnatural" number of clusters for our datasets.

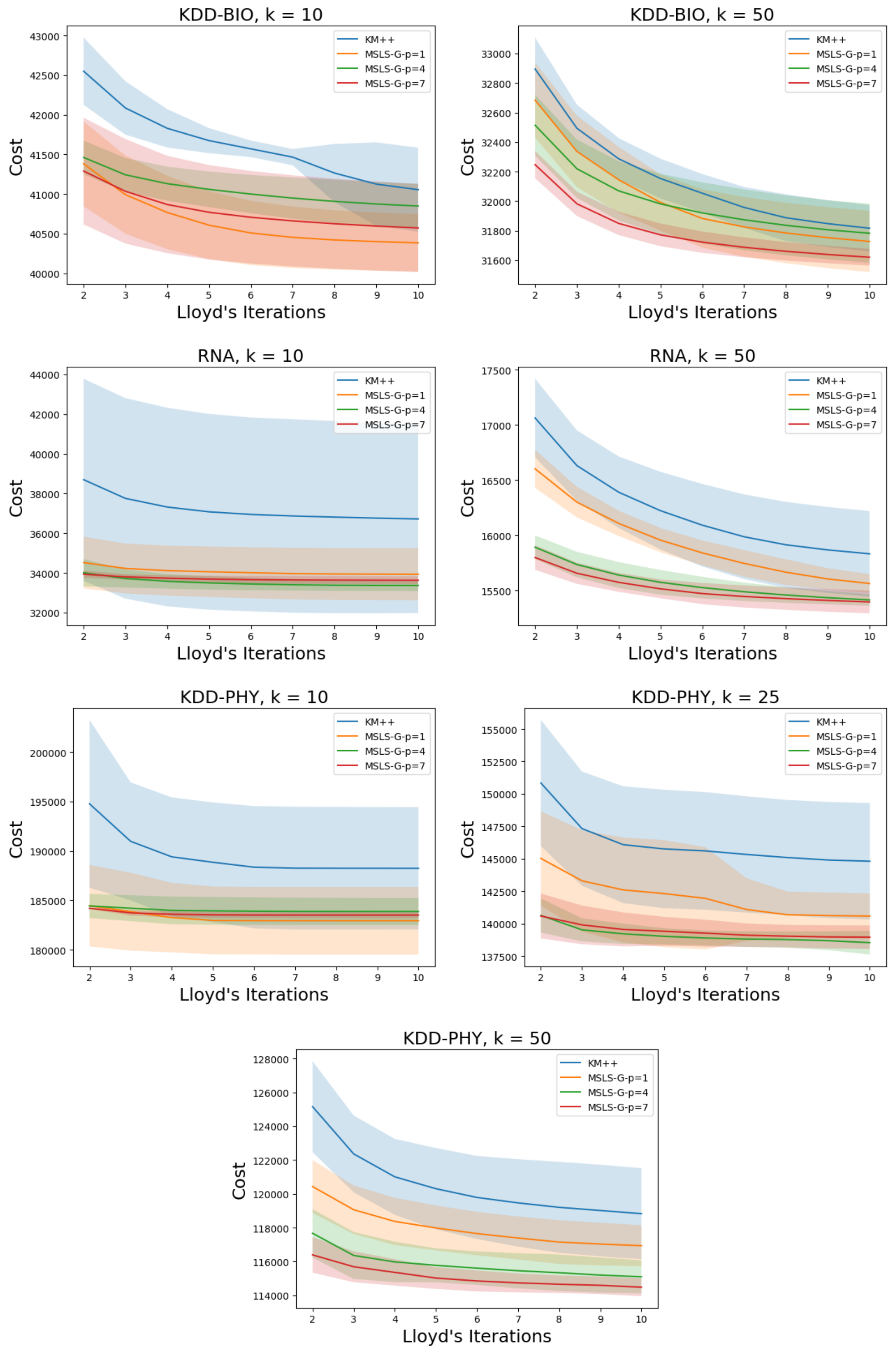


Figure 6: We compare the cost after each of the 10 iterations of Lloyd with seeding from MSLS-G, for $p \in \{1, 4, 7, 10\}$ and 15 local search steps and KM++, for $k \in \{10, 25, 50\}$. We excluded the degenerate case $p = k = 10$, and the experiments reported in the main body of the paper.