Topics in Approximation and Online Algorithms

Guru Guruganesh

CMU-CS-18-121

August 26, 2018

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Anupam Gupta, Chair
R. Ravi
Bernhard Haeupler
Nikhil Bansal

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant numbers CCF-1319811 and CCF-1536002.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

To Mrs. Plachta.

Abstract

Modern computer science is very interested in finding efficient solutions to optimization problems. Many of these problems are NP-Hard and as a result, are unlikely to be solved optimally in a reasonable amount of time. In this context, we need new methods to devise meaningful solutions. Finding algorithms which guarantee an approximate solution to all instances has been a useful way to deal with this intractability. In this thesis, we consider two settings that have received wide interest in the theoretical computer science community: Online and Offline.

Approximation Algorithms (Offline Setting) Since the early 90's, there has been a great deal of success in providing approximate solutions to optimization problem when the entire input is present. In a wide variety of areas such as clustering, network design, vehicle routing and classical graph optimization, the quest for approximation algorithms developed new techniques. Approximation algorithms usually compute a good upper or lower bound (sometimes implicitly) and compare their output locally to these bounds. While this approach yields tight results for many problems, we pick problems in three areas where it falls short: Independent Sets in Sparse Graphs, Aversion K-clustering and Fractionally Sub-additive Network Design. For each of these problems, we achieve the best approximation ratios by using a new analysis which relies on understanding the global structure of the problem.

Online Algorithms Online algorithms have flourished as way to deal with decision making with incomplete information. While the worst-case analysis has been very useful, it often leads to very pessimistic bounds. To deal with this issue, we go beyond the worst-case setting and make conservative assumptions to relax the worst case setting. In this context, we study three classical problems: Online Matroid Intersection, Online Dynamic Bin Packing with recourse, and Smooth Online Convex Optimization. For each of the problem, we keep much of the worst-case competitive ratio model but make natural relaxations to achieve results that are not possible (or conjectured to be impossible) in the worst-case model.

Acknowledgments

My first and foremost acknowledgment is to my advisor Anupam Gupta. For all his patient advice, the endless hours he spent with me on matters technical and otherwise, and for all the things I learned from him during my stay at CMU and continue to do so, I will be forever grateful to Anupam. I am also grateful to my committee members R. Ravi, Bernhard Haeupler and Nikhil Bansal. I have learnt a great deal about doing research from watching and working with them.

Much of the work done in this thesis has been in collaboration with others. I thank Nikhil Bansal, Anupam Gupta, Melanie Schmidt, R. Ravi, Jennifer Igleasias, Laura Sanitá, Sahil Singla, David Wajc, and Amit Kumar for their insights and help with these projects. My time here was made all the more wonderful thanks to the stimulating theory group here. I will forever be grateful to the courses, theory lunches and the reading group sessions that taught me so much. Much of this is made possible by the wonderful staff at CMU and I owe a special thanks to Deb Cavlovich, Nancy Conway and Jenn Landlefield for making my interactions with the bureaucracy so frictionless.

I want to thank my friends who have been with me through the good times and bad: Neil, Manzil, Anirudh, Saransh, Joy, Goran, Jakub, John, Frank, Matthew(s), Ray, Serge and Nicholas. This thesis would not have been possible without the support and dedication of my parents and the unending love of awwa(s), thatha and atha. Finally, I want to thank Mrs. Plachta, for showing me that learning is joyful. This thesis is dedicated to her memory.

Contents

Chapter 1

Introduction

One of the central roles of computer science today is to solve optimization problems efficiently. In a seminal paper, Edmonds [50] advocated that an algorithm is efficient if it runs in time that is polynomial in the input length. This definition serves as a useful proxy for algorithms that can be run in practice. One corollary of this definition is that brute-force solutions, which tend to be exponential in the input length, are inefficient. Cook [42] formalized this by introducing the complexity classes P and NP. Subsequently, Karp [103] showed that a large class of natural problems are NP-Hard, and unlikely to have efficient algorithms.

Since Karp's original discovery, it is now known that many interesting optimization problems are NP-Hard. Assuming the widely held belief that $\mathsf{P} \neq \mathsf{NP}$, it is impossible for an algorithm to run in polynomial time and provide optimal solutions to any instance. One way to circumvent this barrier and provide meaningful solutions is to relax the requirement that the algorithm output an optimal solution; instead, it can give an approximate solution. More formally, this thesis will deal with the following optimization problem:

Definition 1.0.1 *An optimization problem $\mathcal{P}$ is denoted by a feasible set $\mathcal{S} \subseteq \mathbb{R}^d$ and a function $f : \mathcal{S} \to \mathbb{R}_{\geq 0}$. We say that $\mathcal{P}$ is a minimization problem if the algorithm is tasked to find $\text{argmin}_{x \in \mathcal{S}} f(x)$ and $\mathcal{P}$ is a maximization problem if the algorithm is tasked to find $\text{argmin}_{x \in \mathcal{S}} f(x)$.*

We use the following definition of an approximation algorithm, first given in [139].

Definition 1.0.2 *An α-approximation algorithm for an optimization problem $\mathcal{P} = (\mathcal{S}, f)$is a polynomial- time algorithm that for all instances of the problem produces a solution $x \in \mathcal{S}$ whose value is within a factor of α of the value of an optimal solution i.e. $f(x) \geq \alpha \cdot \max_{y \in \mathcal{S}} f(y)$.*

This framework captures a wide variety of problems including many problems in discrete mathematics including matching, clustering, network design, as well as some problems in a continuous setting such as minimizing a convex function.

Offline Algorithms: In the context of approximation algorithms, we assume that the input data is given in full and the function f is usually easy to compute given the input. For example, consider the problem of finding the maximum cut in a graph G. The input is the graph $G = (V, E)$ given as a set of vertices V and edges E. The feasible set $S \in \{0, 1\}^n$ corresponds to the set of indicator vectors for the cuts in G. Clearly, given a point $x \in \mathbb{R}^V$, it is easy to check if it corresponds to a cut and the function f would simply be the count the number of edges in the cut.

For a maximization (resp. minimization) problem, in order to find good approximation algorithms, one needs to come up with good upper-bounds (resp. lower-bounds) for the optimum. A common paradigm to compute such parameters is to use convex programming to solve a relaxed version of the problem. Subsequently, we generate a feasible solution $x \in \mathcal{S}$ and compare it with the convex programming solution. This yields provable guarantees on the approximation algorithm.

Online Algorithms: The assumption that the algorithm always has all of the input before it, is not pragmatic. For example, consider the problem of keeping pages in memory. In this problem, when a page is requested, the algorithm must decide immediately and irrevocably which pages to keep in cache. If the requested page is in the cache, then the algorithm pays nothing but if it is not, then it must pay a cost to put this page into the cache. The algorithm does not know the request sequence in advance and must make a decision *online* as the requests. Furthermore, it is not even clear what the optimal solution in this context means. A beautiful idea introduced by Sleator and Tarjan [134], is competitive analysis: at each time step, the algorithms cost is compared with the worst case cost of the optimal solution assuming all the requests arrived at once. It is surprising that one can compete against an adversary who has more information and no computational constraints, yet for many algorithms competitive analysis has proved to be very effective.

In this thesis, we tackle problems in both online and offline approximation algorithms.

1.1 Approximation Algorithms (Offline Setting)

Since the 90's many optimization problems have been addressed through approximation algorithms. Much of the analysis has been local in nature. Usually, the intended solution is dervied by making small adjustments to the upper-bound (which is often extracted from a convex program) and the analysis compares the intended solution with the upper-bound locally. For the Max-Cut problem, the celebrated result of Goemans and Williamson [69] uses a semidefinite program to generate vectors for each vertex. Then, via a simple hyperplane rounding technique, valid cuts for the graph are generated. To analyze the cost of the cut, the analysis compares the probability of any edge getting cut with the SDP objective generated by the vectors. This analysis is done edge by edge, and is an example of a local analysis. Many techniques which have been developed such as randomized rounding, iterative rounding, primal dual methods, region growing algorithms are all examples of local analysis.

The use of local analysis has been very successful because it is simple and provides tight approximations (assuming P $\neq$ NP). However, there are several important problems for which this analysis falls short. In this thesis, we will look at three such problems and show that we can make progress with a more global analysis. Although these problems span different subdomains they are unified in the property that despite the belief that standard convex relaxations are good upper-bounds (lower-bounds), the best approximation algorithm were substantially worse. The standard techniques to round fail to produce good results as they are local in nature. We will show that a more global approach yields better approximation algorithms.

1.1.1 Independent Sets in Sparse Graphs

In chapter 2, we study the problem of finding independent sets in d-regular graphs using semi-definite programming. This is a fundamental problem in graph theory and finding good algorithms for independent sets has long been a benchmark for new techniques.

The main problem that we address in this chapter is the following:

Problem 1.1.1 *Given a graph G that has maximum degree d, what is the best approximation to the maximum independent set $\alpha(G)$ that can be achieved in polynomial time?*

There is hardness of $d/\log(d)^2$ assuming the Unique Games Conjecture. On the other hand, there is a $d/\log(d)$-approximation algorithm using semi-definite programming. Closing this gap remained an intriguing open question. Our main result answers this question and shows that we can close this gap. We do this by showing that the average size of an independent set is large when the solutions generated by the certain semidefinite programs are large. We give a detailed introduction to the problem in section 2.1.

1.1.2 Aversion K-clustering

Clustering is a natural problem that arises in many contexts. We consider the following problem that arises in the context of computing approximate Nash equilibrium in extensive-formulation games. In this chapter, we consider the following clustering problem: given a metric space (X, D) with a set of clients, and a parameter k, the goal is to group the clients into k clusters. For each client j, the cost incurred by j is its distance to the furthest client in its cluster. The cost of the clustering is the sum over all clients, of the cost incurred by the client. Formally,

Problem 1.1.2 *Given a metric space on a set of clients C, and an integer k, find a partition of $C = \cup_{i=1}^{k} C_i$ so as to minimize $\sum_{i=1}^{k} \sum_{j \in C_i} \max_{\tilde{j} \in C_i} d(j, \tilde{j})$.*

Although it is related in spirit to several other clustering problems, it has some interesting and unique features. Indeed, something that makes this problem difficult is its high "sensitivity" to local perturbations. To overcome this, barrier we use a global partitioning scheme, coupled with the primal-dual algorithm. We give a detailed introduction in section 3.1.

1.1.3 Fractionally Sub-additive Network Design

We study fractionally sub-additive network design, a generalization of existing problems in network design in chapter 4, that we call the *Single-sink fractionally-subadditive network design* problem. In an instance, we are given an undirected graph $G = (V, E)$ with edge costs $w_e \geq 0$ for all $e \in E$, a root node $r \in V$, and k *colors* represented as vertex subsets $C_i \subseteq V \setminus \{r\}$ for all $i \in [k]$, that wish to send flow to r. A feasible solution is an integer capacity installation on the edges of G, such that for every $i \in [k]$, each node in C_i can *simultaneously* send one unit of flow to r. Thus, the total flow sent by color i nodes is is $|C_i|$ while the flows sent from nodes of different colors are instead *non-simultaneous* and can share capacity. An optimal solution is a feasible one that minimizes the total cost of the installation. Formally,

Problem 1.1.3 *Given an undirected graph $G = (V, E)$ with edge costs $w_e \geq 0$, a root node $r \in V$, and k* colors *represented as vertex subsets $C_i \subseteq V \setminus \{r\}$ for all $i \in [k]$, that wish to send flow*

to r. Find the minimum weight set of edges that we must install to ensure that each color C_i can simultaneously *send one unit of flow to r.*

This problem has a standard linear programming relaxation where the demand functions are fractionally sub-additive. However, the best known integrality gap is 2. Due to the complex nature of the demand function traditional techniques such as region growing or even global ones such as iterative rounding fail. We show that we can beat the trivial bound that one achieves by the greedy algorithm for the special case when there are two sets of terminals. This requires a non-local algorithm where the solution is computed by building a global matching. We give a detailed introduction in section 4.1.

1.2 Online Algorithms (Going beyond worst case analysis)

The second part of this thesis deals with online algorithms. Unlike approximation algorithms where we have a complete information, we will have to deal with the inherent loss in not knowing the full information. The standard notion of competitive analysis where one compares to the worst case instance designed by an adversary who is aware of the algorithm. While this has been successful for a certain class of problems, it can be quite pessimistic and produce algorithms that are very complicated. In this regard, it is natural to consider alternative notions where we consider analysis that does not look at the worst case but rather some notion of average case. However, we want to preserve the advantages that come from a worst-case guarantee. In this thesis, we are conservative in the changes to the worst case model that we consider. For each of the problems in consideration, we make the minimal changes to the model to enable us to provide results that are better than a worst case analysis.

1.2.1 Online Matroid Intersection

Online bipartite matching is a fundamental problem that was introduced in the "vertex arrival" model by Karp, Vazirani, and Vazirani [104]. Despite tremendous progress made in the online vertex arrival model, nothing non-trivial was known in the "edge arrival" model where the edges arrive one-by-one. In fact, we consider the more general online matroid intersection problem. The ground elements E of two matroids are presented one-by-one in random order to an online algorithm whose goal is to construct a large common independent set. As the elements arrive, the algorithm must immediately and irrevocably decide whether to pick them, while ensuring that the picked set of elements always form a common independent set. We assume that the algorithm knows the size of E and has access to independence oracles for the already arrived elements. The natural greedy algorithm achieves a competitive ratio of half and it is believed to be tight.

To achieve better results, we relax the model. Instead of considering the model where the matroids and the arrival order is chosen adversarially, we consider the model where the matroids are chosen adversarially but the arrival order is uniformly random. We present the first algorithms that do better than half-competitiveness in this random arrival model. We give a more detailed introduction, along with connections to other problems and works in section 6.1

1.2.2 Online Bin Packing

In chapter 5, we explore a new twist on the classical problem of online bin packing where the algorithm can repack some of the items at some expense.

An instance $\mathcal{I}$ of the bin-packing problem consists of a list of n items of sizes $s_1, s_2, \ldots, s_n \in [0, 1]$. The objective of a bin packing algorithm is to pack the items of $\mathcal{I}$ into a small number of unit-sized bins. Let $OPT(\mathcal{I})$ be the minimum number of unit-sized bins needed to pack all these items. In the online version of the problem, items arrive over time and are immediately and irrevocably packed into bins. In the online setting there is at least a 1.5403-multiplicative gap between the algorithm and OPT in the worst case, even as $OPT \to \infty$.

Once again, these bounds are quite pessimistic and given the wide applicability of the online problem, researchers have considered the problem where a "small" number of repackings is allowed. I.e., how well can we perform if we allow items to be moved between bins when new items arrive and old ones depart? Clearly, some repacking is necessary; to make this question non-trivial, we demand bounded "recourse", i.e., items should be moved sparingly. Formally, a *fully-dynamic* BIN PACKING *algorithm* maintains at every time t, a feasible solution to the BIN PACKING instance $\mathcal{I}_t$ given by items inserted and not yet deleted until time t. Every item i has a size $s_i \in [0, 1]$, and a *movement cost* c_i which the algorithm pays every time item i is moved between bins.

In chapter 5, we consider several recourse models and find the optimal trade-off between the a.c.r. and the worst case recourse. A more detailed introduction can be found in section 5.1.

1.2.3 Smooth Online Convex Optimization

The metrical tasks systems framework introduced by Borodin et al. [28] which captures many online problems such as k-server problem and k-paging problem. In the metrical task system, one is given a finite metric space and a sequence of cost functions. A server can move to any location and pay the cost of the function at that location along with the distance moved. Borodin et al. [28] showed a tight competitive ratio of n where n is the number of points in the metric space. Of course, this is quite pessimistic and this bound could be quite large when the metric space is large but has a lot of structure.

In chapter 7, we study smooth online convex optimization which is a special case of the metrical tasks systems problem. In smooth online convex optimization, the metric space is $\mathbb{R}^d$ where d is small and the cost functions f_i are convex functions. This problem was introduced by Friedman and Linial [64] who considered the special case where f_i are indicators of half-planes and showed that it was sufficient to consider this case. Their explicit goal was to understand the competitive ratio as a function of intrinsic geometric properties of the metric.

In this chapter, we explicitly consider the work-function algorithm for this problem. This algorithm is (or conjectured to be) optimal for various problems including the general metrical task systems problem. Its' behaviour is inherently global as at each step it makes a decision based the global optimum. We analyze various properties of the work function algorithm for the smooth online convex optimization and show that it is competitive when $d = 1$. A more detailed introduction can be found in section 7.1.

Chapter 2

Finding Independent Sets in Degree-d Graphs

2.1 Introduction

Given a graph $G = (V, E)$, an independent set is a subset of vertices S such that no two vertices in S are adjacent. The maximum independent set problem is one of the most well-studied problems in algorithms and graph theory, and its study has led to various remarkable developments such as the seminal result of Lovász [118] in which he introduced the ϑ-function based on semidefinite programming, as well as several surprising results in Ramsey theory and extremal combinatorics.

The main problem that we address in this chapter is the following:

Problem 2.1.1 *Given a graph G that has maximum degree d, what is the best approximation to the maximum independent set $\alpha(G)$ that can be achieved in polynomial time?*

2.2 Related Works

In general graphs, the problem is notoriously hard to approximate. Given a graph on n vertices, the best known algorithm is due to Feige [58], and achieves an approximation ratio of $\widetilde{O}(n/\log^3 n)$; here $\widetilde{O}(\cdot)$ suppresses some $\log\log n$ factors. On the hardness side, a result of Håstad [82] shows that no $n^{1-\varepsilon}$ approximation exists for any constant $\varepsilon > 0$, assuming NP $\not\subseteq$ ZPP. The hardness has been improved more recently to $n/\exp((\log n)^{3/4+\varepsilon})$ by Khot and Ponnuswami [105]. In this chapter, we focus on the case of bounded-degree graphs, with maximum degree d.

Recall that the naïve algorithm (that repeatedly picks an arbitrary vertex v and deletes its neighborhood) produces an independent set of size at least $n/(d+1)$, and hence is a $(d+1)$-approximation. The first $o(d)$-approximation was obtained by Halldórsson and Radhakrishnan [80], who gave a $O(d/\log\log d)$ guarantee, based on a Ramsey theoretic result of Ajtai et al. [3]. Subsequently, an $O(d\,\frac{\log\log d}{\log d})$-approximation was obtained independently by several researchers (see Alon and Kahale [5], Halldórsson [79], Halperin [81]) using the ideas of Karger, Motwani, and Sudan [100] to round the natural SDP for the problem, which was itself based on the Lovász ϑ-function.

On the negative side, Austrin et al. [13] showed an $\Omega(d/\log^2 d)$ hardness of approximation, assuming the Unique Games Conjecture. Assuming P $\neq$ NP, a hardness of $d/\log^4 d$ was recently shown by Chan [32]. We remark that these hardness results only seem to hold when d is a constant or a very mildly increasing function of n. In fact, for $d = \Omega(n)$, the $\Omega(d/\log^2 d)$ hardness of Austrin et al. [13] is inconsistent with the known $O(n/\log^3 n)$ approximation of Feige [58]. Hence throughout this paper, it will be

convenient to view d as being a sufficiently large but fixed constant.

Vishwanathan observed (see [78] for more details) that an $O(d \log\log d/\log d)$ approximation follows from the results of Alon and Kahale [5]. This is the currently best known guarantee. Later, a simpler and direct $O(d \log\log d/\log d)$ was obtained independently by Halperin [81] and Halldórsson [79].

Lower Bounds on the Independence Number. As SDPs can handle cliques, looking at $\vartheta(G)$ naturally leads to Ramsey theoretic considerations. In particular, if $\vartheta(G)$ is small then the trivial $n/(d+1)$ solution already gives a good approximation. Otherwise, if $\vartheta(G)$ is large, then this essentially means that there are no large cliques and one must argue that a large independent set exists (and can be found efficiently).

For bounded degree graphs, a well-known result of this type is that $\alpha(G) = \Omega(n\frac{\log d}{d})$ for triangle-free graphs [2, 132] (i.e. if there are no cliques of size 3). A particularly elegant proof (based on an idea due to Shearer [133]) is in [6]. Moreover this bound is tight, and simple probabilistic constructions show that this bound cannot be improved even for graphs with large girth.

For the case of K_r-free graphs with $r \geq 4$, the situation is less clear. Ajtai et al. [3] showed that K_r-free graphs have $\alpha(G) = \Omega(n(\log(\log d/r))/d)$, which implies that $\alpha(G) = \Omega(n \log\log d/d)$ for $r \ll \log d$. This result was the basis of the $O(d/\log\log d)$ approximation due to [80]. Shearer [133] improved this result substantially and showed that

$$\alpha(G) = \Omega\left(\frac{1}{r}\frac{n}{d}\frac{\log d}{\log\log d}\right)$$

for K_r-free graphs. This result is based on an elegant entropy based approach that has subsequently found many applications. However, it is not known how to make this method algorithmic. This bound still seems far from optimum. In particular, it is possible that the dependence on r could be $\log r$. Note that the above bound is trivial when $r \geq \frac{\log d}{\log\log d}$. For constant r, in particular $r = 4$, it is also an important open question whether the $\log\log d$ factor above can be removed.

Interestingly, Shearer's bound also implies another (non-algorithmic) proof that the SDP has integrality gap $O\Big((d \log\log d)/\log d\Big)$. To see this, suppose the SDP objective is n/r. This essentially implies that the graph is K_r-free as roughly each vertex contributes about $x_i = 1/r$ (formally, one can delete the vertices with $x_i \leq 1/(2r)$ and consider the residual graph). Then the integrality gap is $(n/r)/\alpha(G)$ which by Shearer's bounds is at least $(d \log\log d)/\log d$. It is interesting to note that both Halperin's approach and Shearer's bound seem to get stuck at the same point.

Alon [4] generalized the triangle-free result in a different direction, also using the entropy method. He considered locally k-colorable graphs, where the neighborhood of every vertex is k-colorable and showed that $\alpha(G) = \Omega\left(\frac{n}{d}\frac{\log d}{\log k+1}\right)$. Note that triangle-free graphs are locally 1-

colorable. This result also holds under weaker conditions, and plays a key role in bounding the integrality gap of SA^+ relaxations.

Bounds on the chromatic number. Many of the above results also generalize to the much more demanding setting of list coloring. All of them are based on the "nibble" method, but require increasingly sophisticated ideas. In particular these results give a bound of $\tilde{\Omega}_r(d/\log d)$ on the list chromatic number of K_r-free or locally r-colorable graphs. The intuition for why $O(d/\log d)$ arises can be seen via a coupon-collector argument: if each vertex in the neighborhood $N(v)$ chooses a color from s colors independently and u.a.r., they will use up all s colors unless $d \leq O(s \log s)$, or $s \geq \Omega(d/\log d)$. (Of course, the colors at the neighbors are not chosen uniformly or independently, which substantially complicates the arguments.) Kim showed that $\chi_\ell(G) = O(d/\log d)$ for graphs with girth at least 5 [106]. His idea was that for any v, and $u, w \in N(v)$, $N(u) \cap N(w) = \{v\}$ because of the girth, and hence the available colors at u, w evolve essentially independently, and hence conform to the intuition.

These ideas fail for triangle-free graphs (of girth 4): we could have a vertex v, with $u, w \in N(v)$, and $N(u) = N(w)$ (i.e., all their neighbors are common). In this case the lists of available colors at u and w are far from independent: they would be *completely identical*. Johansson [97] had the crucial insight that this positive correlation is not a problem, since there is no edge between u and w (because of triangle-freeness!). His clever proof introduced the crucial notions of entropy and energy to capture and control the positive correlation along edges in such K_3-free graphs.

If there are triangles, say if the graphs are only locally k-colorable, then using these ideas naïvely fails. Another key idea, also introduced by Johansson [96], is to actually modify the standard nibble process by introducing a *probability reshuffling* step at each vertex depending on its local graph structure, which makes it more complicated. In [19], we give his result for locally-colorable and for K_r-free graphs in its entirety.

2.3 Our Results

The gap between the $\Omega(d/\log^2 d)$-hardness and the $\widetilde{O}(d/\log d)$-approximation arises for the following fundamental reason. Approaches based on the SDP work extremely well if the ϑ-function has value more than $\widetilde{O}(n/\log d)$, but not below this threshold. In order to to show an $\Omega(d/\log d)$-hardness result, at the very least, one needs an instance with SDP value around $n/\log d$, but optimum integral value about n/d. While graphs with the latter property clearly exist (e.g., a graph consisting of $n/(d+1)$ disjoint cliques K_{d+1}), the SDP value for such graphs seems to be low. In particular, having a large SDP value imposes various constraints on the graph (for example, they cannot contain many large cliques) which might allow the optimum to be non-trivially larger than n/d, for example due to Ramsey-theoretic reasons.

Our results resolve some of these questions. Our first result considers the integrality gap of the standard SDP relaxation for independent set (without applying any lift-and-project steps). We show that it is more powerful than the guarantee given by Alon and Kahale [5] and Halperin [81].

Theorem 2.3.1 *On graphs with maximum degree d, the standard ϑ-function-based SDP formulation for the independent set problem has an integrality gap of $\widetilde{O}(d/\log^{3/2} d)$.*

The proof of Theorem 2.3.1 is non-constructive; while it shows that the SDP value is within the claimed factor of the optimal independent set size, it does not give an efficient algorithm to find such an approximate solution. Finding such an algorithm remains an open question.

The main technical ingredient behind Theorem 2.3.1 is the following new Ramsey-type result about the existence of large independent sets in K_r-free graphs. This builds on a long line of previous results in Ramsey theory (some of which we discuss in Section 2.4), and is of independent interest. (Recall that $\alpha(G)$ is the maximum independent set size in G.)

Theorem 2.3.2 *For any $r > 0$, if G is a K_r-free graph with maximum degree d then*

$$\alpha(G) = \Omega\left(\frac{n}{d} \cdot \max\left(\frac{\log d}{r \log\log d}, \left(\frac{\log d}{\log r}\right)^{1/2}\right)\right). \tag{2.1}$$

Previously, the best known bound for K_r-free graphs was $\Omega(\frac{n}{d}\,\frac{\log d}{r\log\log d})$ given by Shearer [133]. Observe the dependence on r: when $r \geq \frac{\log d}{\log\log d}$, i.e., when we are only guaranteed to exclude very large cliques, Shearer's result does not give anything better than the trivial n/d bound. It is in this range of $r \geq \log d$ that the second term in the maximization in (2.1) starts to perform better and give a non-trivial improvement. In particular, if G does not contain cliques of size $r = O(\log^{3/2} d)$ (which will be the interesting case for Theorem 2.3.1), Theorem 2.3.2 gives a bound of $\widetilde{\Omega}(\frac{n}{d}\,(\log d)^{1/2})$. Even for substantially larger values such as $r = \exp(\log^{1-2\varepsilon} d)$, this gives a non-trivial bound of $\widetilde{O}(\frac{n}{d}\,\log^{\varepsilon} d)$.

Improving on Shearer's bound has been a long-standing open problem in the area, and it is conceivable that the right answer for K_r-free graphs of maximum degree d is $\alpha(G) \geq \frac{n}{d}\frac{\log d}{\log r}$. This would be best possible, as we give a simple construction showing an upper bound of $\alpha(G) = O(\frac{n}{d}\frac{\log d}{\log r})$ for $r \geq \log d$, which to the best of our knowledge is the smallest upper bound currently known. The gap between our lower bound and this upper bound remains an intriguing one to close; in fact it follows from our proof of Theorem 2.3.1 that such a lower bound would imply an $\widetilde{O}(d/\log^2 d)$ integrality gap for the standard SDP. Alon [4] shows that this bound is achievable under the stronger condition that the neighborhood of each vertex is $(r-1)$-colorable.

Our proof of theorem 2.3.2 is non-algorithmic and does not give a $\widetilde{O}(d/\log^{3/2} d)$ approximation algorithm. The next set of results consider using lift-and-project techniques to address the approximability of the problem. We consider the standard LP formulation for the independent set problem strengthened by ℓ levels of the Sherali-Adams hierarchy, together with semidefinite constraints at the first level. We will refer to this as ℓ levels of the mixed hierarchy (this is also referred to as the SA^+ hierarchy) and denote this relaxation by $SA^+_{(\ell)}$. Our first result is the following.

Theorem 2.3.3 *The value of the $O(\log^4 d)$-level SA^+ semidefinite relaxation has an integrality gap of $O(d(\log\log d)^2/\log^2 d)$.*

The main observation behind this result is that as the SA^+ relaxation specifies a local distribution on independent sets, and if the relaxation has high objective value then it must be that any polylog(d) size subset of vertices X must contain a large independent subset. We can then use a result of Alon [4], in turn based on the above-mentioned result of Shearer [133], to show that such graphs have non-trivially large independents sets.

Unfortunately, Alon's argument is non-algorithmic; it shows that the lifted SDP has a small integrality gap, but does not give a corresponding approximation algorithm with running time sub-exponential in n. Our next result makes this integrality gap result algorithmic, although at the expense of more levels and a higher running time.

Theorem 2.3.4 *There is an $\widetilde{O}(d/\log^2 d)$-approximation algorithm with running time[1] $\mathrm{poly}(n) \cdot 2^{O(d)}$, based on rounding a d-level SA^+ semidefinite relaxation.*

The improvement is simple, and is based on bringing the right tool to bear on the problem: instead of using the non-constructive argument of Alon [4], we use an ingenious and remarkable (and stronger) result of Johansson [96], who shows that the list-chromatic number of such locally-colorable graphs is $\chi_\ell(G) = O(d\frac{\log k}{\log d})$. His result is based on a very clever application of the Rödl "nibble" method, together with Lovász Local Lemma to tightly control the various parameters of the process at every vertex in the graph. Applying Johansson's result to our problem gives us the desired algorithm.

Finally, the proof of Theorem 2.3.4 also implies the following new results about the LP-based Sherali-Adams (SA) hierarchies, without any SDP constraints.

Corollary 2.3.5 *The LP relaxation with clique constraints on sets of size up to $\log d$ (and hence the relaxation $SA_{(\log d)}$) has an integrality gap of $\widetilde{O}(d/\log d)$. Moreover, the relaxation $SA_{(d)}$ can be used to find an independent set achieving an $\widetilde{O}(d/\log d)$ approximation in time $\mathrm{poly}(n) \cdot 2^{O(d)}$.*

Since LP-based relaxations have traditionally been found to be very weak for the independent set problem, it may be somewhat surprising that a few rounds of the SA-hierarchy improves the integrality gap by a non-trivial amount.

Theorem 2.3.2, our Ramsey-theoretic result, extends to the case when d is the average degree of the graph by first deleting the (at most $n/2$) vertices with degree more than $2\overline{d}$ and then applying the results. We also show that weaker versions of our SDP-based approximation results hold when d is replaced by the average degree instead of the maximum degree. Moreover, we show that the loss in approximation ratio when going from max-degree to average degree is inherent.

2.4 Preliminaries

Given the input graph $G = (V, E)$, we will denote the vertex set V by $[n] = \{1, \ldots, n\}$. Let $\alpha(G)$ denote the size of a maximum independent set in G, and d denote the maximum degree in G. The naïve greedy algorithm implies $\alpha(G) \geq n/(d+1)$ for every G. As the greedy guarantee is tight in general (e.g., if the graph is a disjoint union of $n/(d+1)$ copies of the clique K_{d+1}), the trivial upper bound of $\alpha(G) \leq n$ cannot give an approximation better than $d+1$ and hence stronger upper bounds are needed. A natural bound is the clique-cover number $\overline{\chi}(G)$, defined as the minimum number of vertex-disjoint cliques needed to cover V. As any independent set can contain at most one vertex from any clique, $\alpha(G) \leq \overline{\chi}(G)$.

Standard LP/ SDP Relaxations. In the standard LP relaxation for the independent set problem, there is variable x_i for each vertex i that is intended to be 1 if i lies in the independent set and 0

[1]While a d-level SA^+ relaxation has size $n^{O(d)}$ in general, our relaxation only uses variables corresponding to subsets of vertices that lie in the neighborhood of some vertex v, and thus has $n \cdot 2^{O(d)}$ variables.

otherwise. The LP is the following:

$$\max \sum_i x_i, \quad \text{s.t.} \quad x_i + x_j \leq 1 \quad \forall (i,j) \in E, \quad \text{and} \quad x_i \in [0,1] \quad \forall i \in [n]. \tag{2.2}$$

Observe that this linear program is very weak, and cannot give an approximation better than $(d+1)/2$: even if the graph consists of $n/(d+1)$ copies of K_{d+1}, the solution $x_i = 1/2$ for each i is a feasible one.

In the standard SDP relaxation, there is a special unit vector v_0 (intended to indicate 1) and a vector v_i for each vertex i. The vector v_i is intended to be v_0 if i lies in the independent set and be $\mathbf{0}$ otherwise. This gives the following relaxation:

$$\max \sum_i v_i \cdot v_0, \quad \text{s.t.} \quad v_0 \cdot v_0 = 1, \quad v_0 \cdot v_i = v_i \cdot v_i \quad \forall i \in [n], \quad \text{and} \quad v_i \cdot v_j = 0 \quad \forall (i,j) \in E. \tag{2.3}$$

Let X denote the $(n+1) \times (n+1)$ Gram matrix with entries $x_{ij} = v_i \cdot v_j$, for $i, j \in \{0, \ldots, n\}$. Then we have the equivalent relaxation

$$\max \sum_i x_{0i}, \quad \text{s.t.} \quad x_{00} = 1, \quad x_{0i} = x_{ii} \quad \forall i \in [n], \quad x_{ij} = 0 \quad \forall (i,j) \in E \quad \text{and} \quad X \succeq 0. \tag{2.4}$$

The above SDP, which is equivalent to the well-known ϑ-function of Lovász (see, e.g., [113, Lemma 3.4.4]), satisfies $\alpha(G) \leq \vartheta(G) \leq \overline{\chi}(G)$. The $O(d\frac{\log\log d}{\log d})$ approximations due to [5, 79, 81] are all based on this SDP. Indeed, we use the following important result due to Halperin [81] about the performance of the SDP.

Theorem 2.4.1 (Halperin [81], Lemma 5.2) *Let $\eta \in [0, \frac{1}{2}]$ be a parameter and let Z be the collection of vectors v_i satisfying $\|v_i\|^2 \geq \eta$ in the SDP solution. Then there is an algorithm that returns an independent set of size $\Omega\left(\frac{d^{2\eta}}{d\sqrt{\ln d}}|Z|\right)$.*

(The statement above differs slightly from the one in [81] since Halperin works with a $\{-1, 1\}$ formulation; a proof of its equivalence appears in Appendix A.2). Note that if $\eta = c \log\log d / \log d$, then for $c \leq 1/4$ Theorem 2.4.1 does not return any non-trivial independent set. On the other hand, for $c \geq 1/4$ the size of the independent set returned rises exponentially fast with c.

For more details on SDPs, and the Lovász ϑ-function, we refer the reader to [67, 75].

Lift-and-project Hierarchies. An excellent introduction to hierarchies and their algorithmic uses can be found in [38, 112]. Here, we only describe the most basic facts needed for this paper.

The Sherali-Adams (SA) hierarchy defines a hierarchy of linear programs with increasingly tighter relaxations. At level t, there is a variable Y_S for each subset $S \subseteq [n]$ with $|S| \leq t+1$. Intuitively, one views X_S as the probability that all the variables in S are set to 1. Such a solution can be viewed as specifying a local distribution over valid $\{0,1\}$-solutions for each set S of size at most $t+1$. A formal description of the t-round Sherali-Adams LP $SA_{(t)}$ for the independent set problem can be found in [38, Lemma 1].

More formally, for the independent set problem we have the following theorem from [38].

Theorem 2.4.2 ([38], Lemma 1) *Consider a family of distributions $\{\mathcal{D}(S)\}_{S \subseteq [n]: |S| \leq t+2}$, where each $\mathcal{D}(S)$ is defined over $\{0,1\}^S$. If the distributions satisfy*

1. *For all $(i,j) \in E$ and $S \supseteq \{i,j\}$, it holds that $\Pr_{\mathcal{D}(S)}[(x_i = 1) \cap (x_j = 1)] = 0$, and*
2. *For all $S' \subseteq S \subset [n]$ with $|S| \leq t+1$, the distribution $\mathcal{D}(S'), \mathcal{D}(S)$ agree on S'.*

Then $X_S = \mathrm{P}_{\mathcal{D}(S)}[\wedge_{i \in S}(x_i = 1)]$ is a feasible solution for the level-t Sherali Adams relaxation.

Conversely, for any feasible solution $\{X'_S\}$ for the level-$(t+1)$ Sherali-Adams relaxation, there exists a family of distributions satisfying the above properties, as well as $X_{S'} = \mathrm{P}_{\mathcal{D}(S)}[\wedge_{i \in S'}(x_i = 1)] = X'_{S'}$ for all $S' \subseteq S \subset [n]$ such that $|S| \leq t+1$.

Here, Condition 1 implies that for a subset of vertices S with $|S| \leq t+1$, the local-distribution $\mathcal{D}(S)$ has support on the valid independent sets in the graph induced on S, and Condition 2 guarantees that different local distributions induce a consistent distribution on the common elements.

For our purposes, we will also impose the PSD constraint on the variables x_{ij} at the first level (i.e., we add the constraints in (2.4) on x_{ij} variables). We will call this the t-level SA^+ formulation and denote it by $SA^+_{(t)}$. Such a solution specifies values x_S for multi-sets S with $|S| \leq \ell + 1$. To keep the notation consistent with the LP (2.2), we will use x_i to denote the marginals x_{ii} on singleton vertices.

2.5 Integrality Gap of the Standard SDP

In this section, we show Theorem 2.3.1, that the integrality gap of the standard Lovász ϑ-function based SDP relaxation is

$$O\big(d\big(\tfrac{\log\log d}{\log d}\big)^{3/2}\big) = \widetilde{O}\big(d/\log^{3/2} d\big).$$

To show this we prove the following result (which is Theorem 2.3.2, restated):

Theorem 2.5.1 *Let G be a K_r-free graph with maximum degree d. Then*

$$\alpha(G) = \Omega\left(\frac{n}{d}\max\left(\frac{\log d}{r\log\log d}, \left(\frac{\log d}{\log r}\right)^{1/2}\right)\right).$$

In particular, for $r = \log^c d$ with $c \geq 1$, we get $\alpha(G) = \Omega\big(\frac{n}{d}\big(\frac{\log d}{c\log\log d}\big)^{1/2}\big)$.

We need the following basic facts. The first follows from a simple counting argument (see [4, Lemma 2.2] for a proof).

Lemma 2.5.2 *Let F be a family of $2^{\varepsilon x}$ distinct subsets of an x-element set X. Then the average size of a member of F is at least $\varepsilon x/(10\log(1+1/\varepsilon))$.*

Fact 2.5.3 *Let G be a K_r-free graph on x vertices, then*

$$\alpha(G) \geq \max\left(\frac{x^{1/r}}{2}, \frac{\log x}{\log(2r)}\right).$$

Note that the latter bound is stronger when r is large, i.e., roughly when $r \geq \log x/\log\log x$.

Proof: Let $R(s,t)$ denote the off-diagonal (s,t)-Ramsey number, defined as the smallest number n such that any graph on n vertices contains either an independent set of size s or a clique of size t.

It is well known that $R(s,t) \leq \binom{s+t-2}{s-1}$ [54]. Approximating the binomial gives us the bounds $R(s,t) \leq (2s)^t$ and $R(s,t) \leq (2t)^s$; the former is useful for $t \leq s$ and the latter for $s \leq t$. If

we set $R(s,t) = x$ and $t = r$, the first bound gives $s \geq (1/2)x^{1/r}$ and the second bound gives $s \geq \log x / \log(2r)$. ■

We will be interested in lower bounding the number of independent sets $\mathcal{I}$ in a K_r-free graph. Clearly, $\mathcal{I} \geq 2^{\alpha(G)}$ (consider every subset of a maximum independent set). However the following improved estimate will play a key role in Theorem 2.5.1. Roughly speaking it says that if $\alpha(G)$ is small, in particular of size logarithmic in x, then the independent sets are spread all over G, and hence their number is close to $x^{\Omega(\alpha(G))}$.

Theorem 2.5.4 *Let G be a K_r-free graph on x vertices, and let $\mathcal{I}$ denote the number of independent sets in G. Then we have*

$$\log \mathcal{I} \geq \max\left(\frac{x^{1/r}}{2}, \frac{\log^2 x}{6 \log 2r}\right).$$

Proof: The first bound follows trivially from Fact 2.5.3, and hence we focus on the second bound. Also, assume $r \geq 3$ and $x \geq 64$ else the second bound is trivial.

Define $s := \log x / \log(2r)$. Let G' be the graph obtained by sampling each vertex of G independently with probability $p := 2/x^{1/2}$. The expected number of vertices in G' is $px = 2x^{1/2}$. Let $\mathcal{G}$ denote the good event that G' has at least $x^{1/2}$ vertices. Clearly, $Pr[\mathcal{G}] \geq 1/2$ (in fact it is exponentially close to 1). Since the graph G' is also K_r-free, conditioned on the event $\mathcal{G}$, by Fact 2.5.3 it has an independent set of size at least $\log(x^{1/2})/\log(2r) = s/2$. Thus the expected number of independent sets of size $s/2$ in G' is at least $1/2$.

Now consider some independent set Y of size $s/2$ in G. The probability that Y survives in G' is exactly $p^{s/2}$. As the expected number of independent sets of size $s/2$ in G' is at least $1/2$, it follows that G must contain at least $(1/2)(1/p^{s/2})$ independent sets of $s/2$. This gives us that

$$\log \mathcal{I} \geq \frac{s}{2}\log\left(\frac{1}{p}\right) - 1 \geq \frac{s}{2}\log x^{1/2} - \frac{s}{2} - 1 \geq \frac{s}{6}\log x,$$

where the last inequality assumes that x is large enough. ■

We are now ready to prove Theorem 2.5.1.

Proof: We can assume that $d \geq 16$, else the claim is trivial. Our arguments follow the probabilistic approach of [4, 133]. Let W be a random independent set of vertices in G, chosen uniformly among all independent sets in G. For each vertex v, let X_v be a random variable defined as $X_v = d|\{v\} \cap W| + |N(v) \cap W|$.

Observe that $|W|$ can be written as $\sum_v |v \cap W|$; moreover, it satisfies $|W| \geq (1/d)\sum_v |N(v) \cap W|$, since a vertex in W can be in at most d sets $N(v)$. Hence we have that

$$|W| \geq \frac{1}{2d}\sum_v X_v.$$

Let $\gamma = \max\left(\frac{\log d}{r \log\log d}, \left(\frac{\log d}{\log r}\right)^{1/2}\right)$ denote the improvement factor in Theorem 2.5.1 over the trivial bound of n/d. Thus to show that $\alpha(G)$ is large, it suffices to show that

$$\mathbb{E}[X_v] \geq c\gamma \tag{2.5}$$

for each vertex v and some fixed constant c.

In fact, we show that (2.5) holds for every conditioning of the choice of the independent set in $V - (N(v) \cup \{v\})$. In particular, let H denote the subgraph of G induced on $V - (N(v) \cup \{v\})$. For each possible independent set S in H, we will show that

$$\mathbb{E}[X_v \mid W \cap V(H) = S] \geq c\gamma.$$

Fix a choice of S. Let X denote the non-neighbors of S in $N(v)$, and let $x = |X|$. Let ε be such that $2^{\varepsilon x}$ denotes the number of independent sets in the induced subgraph $G[X]$. Now, conditioning on the intersection $W \cap V(H) = S$, there are precisely $2^{\varepsilon x} + 1$ possibilities for W: one in which $W = S \cup \{v\}$, and $2^{\varepsilon x}$ possibilities in which $v \notin W$ and W is the union of S with an independent set in $G[X]$.

By Lemma 2.5.2, the average size of an independent set in X is at least $\frac{\varepsilon x}{10\log(1/\varepsilon+1)}$ and thus we have that

$$\mathbb{E}[X_v \mid W \cap V(H) = S] \geq d\frac{1}{2^{\varepsilon x}+1} + \frac{\varepsilon x}{10\log(1/\varepsilon+1)}\frac{2^{\varepsilon x}}{2^{\varepsilon x}+1} \tag{2.6}$$

Now, if $2^{\varepsilon x} + 1 \leq \sqrt{d}$, then the first term is at least $\sqrt{d}$, and we've shown (2.5) with room to spare. So we can assume that $\varepsilon x \geq (1/2)\log d$. Moreover, by Theorem 2.5.4,

$$\varepsilon x \geq \max\left(\frac{x^{1/r}}{2}, \frac{\log^2 x}{6\log(2r)}\right)$$

and hence the right hand side in (2.6) is at least

$$\begin{aligned} &\frac{1}{20\log(1/\varepsilon+1)}\max\left(\frac{\log d}{2}, \frac{x^{1/r}}{2}, \frac{\log^2 x}{6\log 2r}\right) \\ &\geq \frac{1}{20\log(x+1)}\max\left(\frac{\log d}{2}, \frac{x^{1/r}}{2}, \frac{\log^2 x}{6\log 2r}\right), \end{aligned} \tag{2.7}$$

where the inequality uses $\varepsilon \geq 1/x$ (since $\varepsilon x \geq (1/2)\log d \geq 1$).

First, let's consider the first two expressions in (2.7). If $x \geq \log^r d$, then as $x^{1/r}/\log(x+1)$ is increasing in x, it follows that the right hand side of (2.7) is at least

$$\frac{x^{1/r}}{40\log(x+1)} = \Omega\left(\frac{\log d}{r\log\log d}\right).$$

On the other hand if $x \leq \log^r d$, then we have that the right hand side is again at least

$$\frac{1}{20\log(x+1)}\frac{\log d}{2} = \Omega\left(\frac{\log d}{r\log\log d}\right).$$

Now, consider the first and third expressions in (2.7). Using the fact that $\max(a,b) \geq \sqrt{ab}$ with $a = (\log d)/2$ and $b = (\log^2 x)/(6\log 2r)$, we get that (2.7) is at least $\Omega\left(\frac{\log d}{\log r}\right)^{1/2}$. Hence, for every value of x we get that (2.7) is at least $\Omega(\gamma)$ as desired in (2.5); this completes the proof of Theorem 2.5.1. ■

We can now show the main result of this section.

Theorem 2.5.5 *The standard SDP for independent set has an integrality gap of*

$$O\left(d\left(\frac{\log\log d}{\log d}\right)^{3/2}\right).$$

Proof: Given a graph G on n vertices, let $\beta \in [0,1]$ be such that the SDP on G has objective value βn. If $\beta \leq 2/\log^{3/2} d$, the naïve greedy algorithm already implies a $d/\log^{3/2} d$ approximation. Thus, we will assume that $\beta \geq 2/\log^{3/2} d$. Recall we use x_i to denote the marginals x_{ii} on singleton vertices in the SDP.

Let us delete all the vertices that contribute $x_i \leq \beta/2$ to the objective. The residual graph has objective value at least $\beta n - (\beta/2)n = \beta n/2$.

Let $\eta = 2\log\log d/\log d$. If there are more than $n/\log^2 d$ vertices with $x_i \geq \eta$, applying Theorem 2.4.1 to the collection of these vertices already gives independent set of size at least

$$\Omega\left(\frac{d^{2\eta}}{d\sqrt{\ln d}}\cdot\frac{n}{\log^2 d}\right) = \Omega\left(\frac{n\log^{3/2} d}{d}\right),$$

and hence a $O(d/\log^{3/2} d)$ approximation.

Thus we can assume that fewer than $n/\log^2 d$ vertices have $x_i \geq \eta$. As each vertex can contribute at most 1 to the objective, the SDP objective on the residual graph obtained by deleting the vertices with $x_i \geq \eta$ is at least $\beta n/2 - n/(\log^2 d)$ which is at least $\beta n/3$, since $\beta \geq 2/\log^{3/2} d$.

So we have a feasible SDP solution on a subgraph G' of G, where the objective is at least $\beta n/3$ (here n is the number of vertices in G and not G') and each surviving vertex i has value x_i in the range $[\beta/2, \eta]$.

As $x_i \leq \eta$ for each i, and the SDP objective is at least $\beta n/3$, the number of vertices n' in G' satisfies $n' \geq (\beta n/3)/\eta = \Omega(n\beta/\eta)$. Moreover, as $x_i \geq \beta/2$ for each vertex $i \in G'$, and the SDP does not put more than one unit of probability mass on any clique, it follows that G' is K_r-free for $r = 2/\beta = \log^{3/2} d$. Applying Theorem 2.5.1 to G' with parameter $r = \log^{3/2} d$, we obtain that G' has an independent set of size

$$\Omega\left(\frac{n'}{d}\sqrt{\frac{\log d}{\log r}}\right) = \Omega\left(\frac{n'}{d}\sqrt{\frac{\log d}{\log\log d}}\right) = \Omega\left(\frac{n\,\beta}{d\,\eta}\sqrt{1/\eta}\right) = \Omega\left(\frac{\beta n}{d}\cdot\eta^{-3/2}\right).$$

The SDP objective for G was βn, so the integrality gap is $O(d\eta^{3/2}) = O(d(\frac{\log\log d}{\log d})^{3/2})$. ■

2.5.1 An upper bound

Lemma 2.5.6 *There exists K_r-free graphs G with maximum degree d such that $\alpha(G) \leq O\left(\frac{n}{d}\frac{\log d}{\log r}\right)$ whenever $r \geq \log d$.*

Proof: We use the standard lower bound $R(s,t) = \Omega((\frac{t}{\log t})^{s/2})$ for off-diagonal Ramsey numbers for $t \geq s$. While stronger lower bounds exist (see Theorem 1.2 in [27]), this one suffices for the

lemma. Setting $t = r$ and $s = O\Big(\log d/(\log r - \log\log r)\Big)$, it follows that there exist K_r-free graphs H on d vertices such that $\alpha(H) = O\Big(\log d/(\log r - \log\log r)\Big)$ whenever $t \geq s$. We now apply two simplifications. First, as $\log d \geq s$, we can simplify the condition $t \geq s$ to be $r \geq \log d$. Second, as $\log r \geq 2\log\log r$ for all $r > 1$, we can say $\alpha(H) = O\Big(\frac{\log d}{\log r}\Big)$. Setting G to be n/d disjoint copies of H completes the lemma. ■

2.6 Lift-and-Project Algorithms

2.6.1 Sherali-Adams+ based guarantees

In this section, we give our results bounding the integrality gap of the SA^+ mixed hierarchy. We first show that the $O(\log^4 d)$-level SA^+ relaxation has an integrality gap of $\tilde{O}(d/\log^2 d)$. This result, however, does not give an effective procedure to find such a large independent set. Then we show how to round a vector solution to the d-level SA^+ relaxation to get an independent set of size at least a $\tilde{O}(d/\log^2 d)$ factor of the optimal independent set.

Consider the $SA^+_{(t)}$ relaxation on G; for the subsequent results we choose the value of t to be $O(\log^4 d)$ and d respectively. Let $\mathsf{sdp}_t(G)$ denote its value. We can assume that

$$\mathsf{sdp}_t(G) \geq n/\log^2 d, \tag{2.8}$$

otherwise the naïve greedy algorithm already gives a $O(d/\log^2 d)$ approximation.

Let $\eta = 3\log\log d/\log d$, and Z denote the set of vertices i with $x_i \geq \eta$. We can assume that $|Z| \leq n/(4\log^2 d)$, otherwise applying Theorem 2.4.1 gives an independent set of size $\Omega(|Z| \cdot d^{2\eta}/(d\sqrt{\log d})) = \Omega(n\log^2 d/d)$. Note that we can apply Theorem 2.4.1, since our solution belongs to SA^+ and hence is a valid SDP solution. Hence,

$$\mathsf{sdp}_t(G) \leq |Z| \cdot 1 + (n - |Z|) \cdot \eta \leq (n/(4\log^2 d)) \cdot 1 + n \cdot \eta \leq 2\eta n.$$

Let V' denote the set of vertices i with $x_i \in [1/(4\log^2 d), \eta]$, and let $G' = G[V']$ be the graph induced on these vertices.

Claim 2.6.1 $|V'| \geq \mathsf{sdp}_t(G)/(2\eta)$.

Proof: The total contribution to $\mathsf{sdp}_t(G)$ of vertices i with $x_i \leq 1/(4\log^2 d)$ can be at most $n/(4\log^2 d)$, which by (2.8) is at most $\mathsf{sdp}_t(G)/4$. Similarly, the contribution of vertices in Z is at most $|Z|$, which is again at most $\mathsf{sdp}_t(G)/4$. Together this gives $\mathsf{sdp}_t(G') \geq \mathsf{sdp}_t(G)/2$. As each vertex in V' has $x_i \leq \eta$, the claim follows. ■

Lemma 2.6.2 *Let $t \geq \log^4 d$. For the graph $G' = G[V']$, let $v \in V'$ and let $T \subseteq N_{G'}(v)$ be a subset of neighbors of v in G' having size at most t.*

(a) T contains an independent set of size at least $|T|/\log^2 d$.

(b) the induced subgraph $G'[T]$ is $O(\log^3 d)$-colorable.

Proof: Consider the solution $SA^+_{(t)}$ restricted to G'. Since $|T| \leq t$ and $x_i \geq 1/(4\log^2 d)$ for all $i \in N_{G'}(v)$, we use Theorem 2.4.2 to deduce that the $SA^+_{(t)}$ solution defines a "local distribution" $\{X_S\}_{S \subseteq T}$ over subsets of T with the following properties:

(i) $X_S \geq 0$ and $\sum_{S \subseteq T} X_S = 1$,

(ii) $X_S > 0$ only if S is independent in the induced subgraph $G'[T]$ (and hence in G), and

(iii) for each vertex $i \in T$, it holds that

$$x_i = \sum_{S \subseteq T : i \in S} x_S \geq 1/(4 \log^2 d).$$

Now scaling up the solution $\{X_S\}$ by $4 \log^2 d$ gives a valid fractional coloring of T using $4 \log^2 d$ colors. This means at least one of the color classes must have size at least $|T|/(4 \log^2 d)$. This proves (a).

To prove (b), we can use a set-covering argument. The fractional coloring can be viewed as a fractional set cover of T, where the sets are all independent sets in G. The (fractional) number of sets used is $4 \log^2 d$. Now the integrality gap of the LP relaxation of set cover implies that we can cover T using at most $4 \log^2 d \cdot O(\log |T|) = O(\log^3 d)$. ■

The Integrality Gap Result

We now bound the integrality gap of the $O(\log^4 d)$-round SA^+ relaxation. The following Ramsay-theoretic result will be crucial.

Theorem 2.6.3 (Alon [4], Theorem 1.1) *Let $H = (V, E)$ be a graph on n vertices with maximum degree $d \geq 1$ such that for every vertex $v \in V$ the induced subgraph on the set of all neighbors of v is k-colorable. Then,*

$$\alpha(H) \geq \Omega\left(\frac{n}{d} \frac{\log d}{\log(k+1)}\right).$$

To reduce the number of rounds of SA^+, we use a version of the result above that holds under a considerably weaker condition.

Theorem 2.6.4 (Alon [4]) *Let $H = (V, E)$ be a graph on n vertices with maximum degree d, and let $k \geq 1$ be an integer. If for every vertex v and every subset $T \subset N(v)$ with $|T| \leq k \log^2 d$, it holds that the subgraph induced on T has an independent set of size at least $|T|/k$, then*

$$\alpha(H) \geq \Omega\left(\frac{n}{d} \frac{\log d}{\log(k+1)}\right).$$

For completeness, a proof of Theorem 2.6.4 can be found in Appendix A.2.2.

Now consider the graph $G' = G[V']$. By Lemma 2.6.2(a) with the parameter $t = \log^2 d$, the graph G' satisfies the requirements in Theorem 2.6.4 with $k = \log^2 d$; that theorem gives us that

$$\alpha(G') \geq \Omega\left(\frac{|V'|}{d} \frac{\log d}{\log(\log^2 d)}\right).$$

Finally, using Claim 2.6.1, the integrality gap is

$$\frac{\mathsf{sdp}_{(\log^4 d)}(G)}{\alpha(G)} \leq \frac{\mathsf{sdp}_{(\log^4 d)}(G)}{\alpha(G')} \leq O\left(\frac{d\eta \log(\log^2 d)}{\log d}\right) = \widetilde{O}\left(\frac{d}{\log^2 d}\right),$$

This completes the proof of Theorem 2.3.3.

The Algorithmic Result

To get an algorithm, note that Lemma 2.6.2(b) with $t = d$ implies that the neighborhood of each vertex in G' is $O(\log^3 d)$ colorable. In other words, G' is locally k-colorable for $k = O(\log^3 d)$. We now use Johansson's coloring algorithm for locally k-colorable graphs (Theorem A.1.1) to find an independent set of G' with size

$$\mathsf{alg}(G') = \Omega\left(\frac{|V'|}{d} \cdot \frac{\log d}{\log(k+1)}\right).$$

Using $k = O(\log^3 d)$ and Claim 2.6.1 this implies an algorithm to find independent sets in degree d graphs, with an integrality gap of

$$\frac{\mathsf{sdp}_d(G)}{\mathsf{alg}(G)} \leq \frac{\mathsf{sdp}_d(G)}{\mathsf{alg}(G')} \leq O\left(\frac{d\eta \log(k+1)}{\log d}\right) = \widetilde{O}\left(\frac{d}{\log^2 d}\right).$$

Our algorithm only required a fractional coloring on the neighborhood of vertices. Since they are at most 2^d independent sets in each neighborhood, there are at most $n \cdot 2^d$ relevant variables in our SDP. Hence, we can compute the relevant fractional coloring in time $\mathrm{poly}(n) \cdot 2^{O(d)}$. This completes the proof of Theorem 2.3.4.

2.6.2 Sherali-Adams based guarantees

We prove Corollary 2.3.5, showing the integrality gap of the Sherali-Adams hierarchy (without the SDP constraints).

Proof: Consider the standard LP (2.2) strengthened by the clique inequalities $\sum_{i \in C} x_i \leq 1$ for each clique C with $|C| \leq \log d$. As each clique lies in the neighborhood of some vertex, the number of such cliques is at most $n \cdot \binom{d}{\log d}$. Let βn denote the objective value of this LP relaxation. We assume that $\beta \geq 2/\log d$, otherwise the naïve algorithm already gives a $d/\log d$ approximation.

Let B_0 denote the set of vertices with $x_i \leq 1/\log d = \beta/2$. For $j = 1, \ldots, k$, where $k = \log\log d$, let B_j denote the set of vertices with $x_i \in (2^{j-1}/\log d, 2^j/\log d]$. Note that $\sum_{j \geq 1} \sum_{i \in B_j} x_i = \beta n - \sum_{i \in B_0} x_i \geq \beta n/2$, and thus there exists some index j such that $\sum_{i \in B_j} x_i \geq \beta n/(2k)$.

Let $\gamma = 2^{j-1}/\log d$; for each $i \in B_j$, $x_i \in (\gamma, 2\gamma]$. Since $x_i > \gamma$ for each $i \in B_j$, the clique constraints ensure that the graph induced on B_j is K_r-free for $r = 1/\gamma$. Moreover, since $x_i \leq 2\gamma$ for each $i \in B_j$, $|B_j| \geq \frac{1}{2\gamma} \cdot \frac{\beta n}{2k}$. By Shearer's result for K_r-free graphs we obtain

$$\alpha(B_j) = \Omega\left(|B_j| \cdot \frac{\gamma \log d}{d \log\log d}\right) = \Omega\left(\frac{\beta n \log d}{d(\log\log d)^2}\right).$$

This implies the claim about the integrality gap.

A similar argument implies the constructive result. Let βn denote the value of the $SA_{(d)}$ relaxation. As before, we assume that $\beta \geq 2/\log d$ and divide the vertices into $1 + \log\log d$ classes. Consider the class B_j with $j \geq 1$ that contributes most to the objective, and use the fact that the graph induced on B_j is locally k-colorable for $k = (\log d/2^{j-1} \cdot \log d) = O(\log^2 d)$. As in Section 2.6, we can now use Johansson's coloring algorithm Theorem A.1.1 to find a large independent set. ■

2.7 Open Problems

The main open question from the above work is the resolution of the following open problem:

Problem 2.7.1 *For any K_r-free graph G, does there exist an independent set satisfying $\alpha(G) \geq O(\frac{n \log d}{d \log r})$*

which would imply that the Lovász ϑ-function achieves a $O(\frac{d}{\log^2 d})$ approximation. One of the key bottlenecks in extending this result is to calculate the average size of an independent set. Recently, Feige and Kenyon [60] have obtained better bounds for such estimates in random graphs.

A second important question is making Johansson's methods algorithmic. In particular,

Problem 2.7.2 *Does there exist a polynomial time algorithm to find independent sets in K_r-free graphs when $r \in \Omega(1)$?*

There has been some recent progress on this topic. In particular, Molloy [123] has shown simpler proofs for several of Johansson's results which may be of use in finding better bounds.

Another problem with a similar gap in our understanding is the hypergraph matching problem where there is a $\Omega(\frac{k}{\log k})$ hardness result due Hazan et al. [83], and an upper-bound of $\frac{k+1}{3}$ due to Cygan [44]. A natural open question is :

Problem 2.7.3 *What is the integrality gap of the Lovász ϑ-function for the hypergraph matching problem?*

A more general research direction is understanding the power of hiearchies in finding independent sets in graphs. We know that there are graphs where ϑ-function cannot achieve an aprximation factor better than $n/2^{\sqrt{\log n}}$ due to the work of Feige [57]. However, it is possible that higher rounds of the SOS-hierarchy will yield improved approximation algorithms. In particular, it remains an open question whether these bad examples survive many rounds of SOS hierarchy.

A lot of early work on finding independent sets was motivated by the following interesting problem: Given a 3-colorable graph, what is the minimum number of colors that can be used to give a valid coloring in polynomial time? All known algorithms for this problem color the graphs by finding large independent sets. While the best upper-bounds are polynomial in n, the strongest hardness results are that it is NP-Hard to color them with 4 colors. Assuming a stronger variant of the UGC, it is known to be NP-Hard to color them with $\omega(1)$ colors.

What is the benefit of hiearchies in resolving the difference between coloring vs independent sets? While the two problems, are quite different, it is unclear if their hardness can be seperated. To the best of our knowledge, all algorithms to find a good coloring always do so by repeatedly extracting large independent sets. It remains an open question if they can be made open. Secondly, recetly SDP's have been used to find planted colorings as in the work of Feige. Can we do better using rounds of hierarchies?

Chapter 3

Local k-median

3.1 Introduction

In this chapter, we consider the following clustering problem: given a metric space (X, D) with a set of clients, and a parameter k, the goal is to group the clients into k clusters. For each client j, the cost incurred by j is its distance to the furthest client in its cluster. The cost of the clustering is the sum over all clients, of the cost incurred by the client. We call this problem the *aversion k-clustering problem*. Formally,

Problem 3.1.1 *Given a metric space on a set of clients C, and an integer k, find a partition of $C = \cup_{i=1}^{k} C_i$ so as to minimize $\sum_{i=1}^{k} \sum_{j \in C_i} \max_{\tilde{j} \in C_i} d(j, \tilde{j})$.*

This question is motivated by a problem in developing abstractions of extensive-form games. Since finding equilibria in large extensive form games is computationally expensive, one appealing approach if speeding things up is to develop an abstraction of this game. Since the abstraction is typically much smaller, existing algorithms can be used to solve them to find the optimal strategies, which can be mapped back to the original game. However, there is often some loss in going to the abstraction. Recent work of Kroer and Sandholm [111] on automated abstraction algorithms proposed the following way to model this: since several states of the original game may be collapsed into a single state in the abstraction, the loss for each original state is its distance (in a suitably defined metric) to the furthest state that is collapsed with it. The overall loss is the sum of per-state losses. This is precisely the aversion k-clustering problem we study in this chapter.

To the best of our knowledge, no prior approximation algorithms were known for the aversion k-clustering problem. Although it is related in spirit to several other clustering problems, it has some interesting and unique features. Indeed, something that makes this problem difficult is its high "sensitivity". To explain this, observe that in problems like k-median, if we re-assign a single client j to a new cluster C, loosely this changes the cost by the distance of the client to the new cluster. However, in aversion k-clustering reassigning client j to a new cluster C may also significantly change the cost of all other clients in C, since j may become their new furthest client. This creates problems for most standard techniques used for facilty-location problems. Since our objective is not linear (due to each client paying the distance to its furthest cluster-mate), we cannot even use tree embeddings.

3.2 Related Works

Approximation algorithms for facility location problems have been studied for a long time. Indeed, many approximation techniques have been developed while investigating these problems (see [139]). The problem closest to the *local k-median problem* is naturally the *metric k-median* problem. The first constant-factor for this problem was due to Charikar et al. [35] via rounding the LP; subsequently, primal-dual algorithms were given by Jain and Vazirani [93] and Charikar and Guha [33], a local-search algorithm was given by Arya et al. [12]. The recent approach of Li and Svensson [116] gave a $2.73 + \varepsilon$-approximation, which was improved to $2.675 + \varepsilon$ by Byrka et al. [30]. The current NP-hardness is a $1 + 2/e$-factor due to Jain et al. [94].

The k-median problem sums over each cluster, the sum of distances of clients to their cluster center. Instead of taking the sum of distances within each cluster, we could take the maximum distance within each cluster; this gives the *sum of cluster diameters* problem, for which a $O(1)$-factor is due to Charikar and Panigrahy [34]. And instead of summing diameters over the clusters, if we take the maximum diameter over all clusters, we get the *k-center* problem, for which a 2-approximation is due to Gonzalez [71], and Hochbaum and Shmoys [87], and a matching hardness is due to Hsu and Nemhauser [89].

Another related problem is the *min-sum clustering problem*, where we sum over the clusters of the distances between all pairs within the cluster. Bartal et al. [22] give a $O(\varepsilon^{-1} \log^{1+\varepsilon} n)$-approximation, which was recently improved to $O(\log n)$ by Behsaz et al. [23]. There are easy examples where these problems differ from aversion k-clustering by arbitrarily large factors. Moreover, the non-linearity of our objective function means that we cannot use tree embedding results to get a logarithmic approximation.

Our algorithm takes a primal-dual approach pioneered by Jain and Vazirani [93]; while solving the Lagrangian relaxation and getting a Lagrangian-multiplier preserving algorithm follows relatively easily, the main contribution is in the non-local rounding algorithm. This adds to the body of work exploring such primal-dual techniques, which include the work of Charikar and Panigrahy [34] to give a $O(1)$-factor approximation for the sum of cluster diameters, and Chuzhoy and Rabani [40] in their $O(1)$-factor bicriteria approximation for the capacitated k-median problem. To the best of our understanding, our rounding technique is different from these previous works. Non-local roundings of a different flavor were also recently used for the capacitated k-center problem by Cygan et al. [45] and An et al. [7].

3.3 Our Results and Techniques

Our main result is the following:

Theorem 3.3.1 *There is a constant-factor algorithm for the aversion k-clustering problem.*

A few words about our techniques. To solve this, we first move to a related problem that is more convenient to deal with: in the *local k-median problem*, each potential facility location in the metric space has a "range" R_i associated with it. Like in k-median, we need to open k facilities, to minimize the sum of distances from clients to their assigned facilities. However, we now additionally require that each client j is assigned to some facility i at distance at most R_i. This problem

is NP-hard to approximate, but for our purpose it is sufficient to solve the relaxed version where clients can still connect at distance $\mathcal{O}(\mathsf{R}_i)$.

Again, the (relaxed) locality restriction causes many of the standard techniques for k-median, like local-search and LP-rounding, not to extend to this problem. However, we are successful in extending a primal-dual technique to this problem. The following theorem is our main technical result, from which Theorem 3.3.1 follows immediately.

Theorem 3.3.2 *There is a constant-factor approximation algorithm for the local k-median problem which violates the locality constraints by a constant factor for instances that arise from aversion k-clustering problem.*

We use the primal-dual framework of Jain and Vazirani: we find two solutions that open k_1 and k_2 facilities (such that $k_1 < k < k_2$) such that the "average" of these two solutions has low cost and opens k facilities. We can view this average solution as a "well-behaved" LP solution, which we now have to round to integrally open k facilities.

The main problem with this rounding is the locality constraint — typical algorithms tend to round some fractional facility up to 1, round down close-by fractional facilities to zero to maintain the total facility mass at k, and reroute clients to the newly opened facility without increasing the cost by much. However, the locality constraint in our problem means that such simple rounding approaches fail. For example, the facility that we open may have a very small R_i value, and can only serve clients that are very close to it. However, the clients who want to be rerouted may be too far from this facility to satisfy the locality constraint, even if it is relaxed to $\gamma\mathsf{R}_i$ for some constant γ.

Our main technical contribution, and the novel ingredient of our rounding algorithm is a *non-local rounding approach*. We first transform the fractional solution so that its support is a tree. (Technically, we also require that this tree has some additional properties.) Then we partition this tree into carefully chosen subtrees, so that all the clients in each particular subtree can be reassigned simultaneously without violating the locality. Now choosing the least expensive of these forests to reassign gives us a solution with k facilities and a constant-factor approximation. We feel that this non-local rounding will be useful in other contexts, and hence be of independent interest.

3.4 Solving aversion k-clustering problem via the local k-median problem

3.4.1 Preliminaries

Let (X, D) be a metric space and let $C \subseteq X$ be the set of *clients* and $F \subseteq X$ be the set of *facilities*. The *aversion k-clustering problem* is the task to partition C into a collection $\mathcal{C}$ of k disjoint subsets $C_1, \ldots, C_k$ with $C = \cup_{i=1}^{k} C_i$ such that

$$c_a(\mathcal{C}) := \sum_{\ell=1}^{k} \sum_{j_1 \in C_\ell} \max_{j_2 \in C_\ell} D(j_1, j_2) \tag{P1}$$

is minimized.

For the *local k-median problem*, we additionally get a *radius* (or *range*) R_i for every $i \in F$. We seek a set $\mathcal{F} \subseteq F$ of k facilities that minimizes

$$c_l(\mathcal{F}) := \sum_{j \in C} \min_{\substack{i \in \mathcal{F}, \\ D(i,j) \leq \mathsf{R}_i}} D(i,j).$$

This differs from the classical k-median problem in that a client can only be assigned to a facility if it lies within the facility's radius. It is possible that there is no set of k facilities which can service all clients. If this is the case, we define the minimum clustering cost as infinity. In the following claim, we show that it is NP-hard to decide whether we are in this case or not.

Claim 3.4.1 *Deciding feasibility of a local k-median instance is NP-hard.*

Proof: We use a well-known reduction from set cover. Let $\mathcal{S}$ be a set of sets over a universe $\mathcal{U}$. We construct a metric space that contains a facility i_S for every set $S \in \mathcal{S}$ and a client j_u for every element $u \in \mathcal{U}$. The distance between j_u and i_S is one if $u \in S$ and two otherwise. Observe that this is a metric. We set the radius R_{i_S} of all facilities to one. Observe that there is a feasible solution for this local k-median instance if and only if the set cover instance has a solution with at most k sets. Since deciding whether a set cover instance has a solution with at most k sets is NP-hard [102], it is also NP-hard to decide whether there is a feasible solution for the local k-median problem. ■

Any approximation algorithm has to decide whether there is a feasible solution or not. Hence, we allow the locality constraint to be violated; i.e. a client may connect to a facility i if it is within a radius of $\gamma \mathsf{R}_i$ for a constant γ. We say a solution is a (γ, ψ) bicriteria solution if the solution violates the locality constraints by a factor of γ and has cost at most ψ times the optimal (with respect to the original problem).

3.4.2 Reductions

We show that the aversion k-clustering problem can be reduced to the local k-median problem by sacrificing a constant factor. The idea is to identify a cluster C_ℓ with a pair of points with largest distance and to use this information to represent clusters by an artificial facility with appropriate radius. More precisely, we define the following metric space. Set $F := \{p_{j_1 j_2} \mid j_1, j_2 \in C\}$ and refer to $p_{j_1 j_2}$ as the *midpoint* of clients j_1 and j_2. To extend D from C to $C \cup F$, we set

$$D(j_1, p_{j_1 j_2}) := D(j_2, p_{j_1 j_2}) = \frac{D(j_1, j_2)}{2} \text{ and } D(p_{j_1 j_2}, p_{j_1 j_2}) := 0$$

for all $j_1, j_2 \in C$. So far, no metric property is violated. Now imagine the incompletely defined metric as a weighted undirected graph G on the vertices $C \cup F$ where some edges are missing. Let D be defined as the shortest path metric in G. This is a metric by definition. It coincides with the previously defined distances since in a metric, the direct edge must be a shortest path. For the missing edges, we get that

$$D(j, p_{j_1 j_2}) = D(p_{j_1 j_2}, j) = \min\{D(j, j_1), D(j, j_2)\} + \frac{D(j_1, j_2)}{2} \tag{3.1}$$

for all $j \in C$: The point $p_{j_1 j_2}$ is only connected to j_1 and j_2, thus any path between j and $p_{j_1 j_2}$ has to travel over one of them. Since the edge lengths form a metric, the direct connection between j and j_1 or j_2 is shortest, so either $(j, j_1), (j_1, p_{j_1 j_2})$ or $(j, j_2), (j_2, p_{j_1 j_2})$ is a shortest path. Analogously, we get that

$$D(p_{j_1 j_2}, p_{j_3 j_4}) = D(j_1, j_2)/2 + D(j_3, j_4)/2 + \min\{D(j_1, j_3), D(j_1, j_4), D(j_2, j_3), D(j_2, j_4)\}$$

for all $j_1, j_2, j_3, j_4 \in C$. Finally, we define

$$\mathsf{R}_{p_{j_1 j_2}} := D(j_1, j_2)/2 \tag{3.2}$$

for all $p_{j_1 j_2} \in F$. Notice that our definition of F allows that $j_1 = j_2$. This ensures that singleton clusters can be expressed. Furthermore, notice that $\mathsf{R}_{p_{j_1 j_1}} = 0$, so the facility $p_{j_1 j_1}$ can only serve j_1 (or clients at the same location).

For any facility p_{j_1, j_2}, we will drop the reference to j_1 and j_2 when it is clear from the context. Hence, $p \in F$ refers to the midpoint of some two clients j_1 and j_2 and the radius of the facility R_p simply refers to half the distance between these points. Intuitively, each new "facility" corresponds to a midpoint of two clients in the original problem. These midpoints allow us to cast the current problem as a k-median problem with the addition of locality constraints placed on each facility.

We now show how solutions for the aversion k-clustering problem and (γ, α) bicriteria solutions for the local k-median problem are related. For a client j, let $F_j^\gamma := \{i \in F \mid D(i, j) \leq \gamma \mathsf{R}_i\}$ be the set of facilities that j is allowed to connect to. The following integer linear program (ILP) which is a (natural) modification of the ILP proposed in [15] minimizes over all feasible (γ, α) bicriteria solutions.

$$\begin{aligned}
\min \sum_{j \in C} \sum_{i \in F_j^\gamma} D(i, j) \cdot x_{i,j} & & \text{(ILP}^\gamma\text{)} \\
\sum_{i \in F} y_i \leq k & & \\
\sum_{i \in F_j^\gamma} x_{i,j} \geq 1 & \qquad \forall j \in C & \\
y_i - x_{i,j} \geq 0 & \qquad \forall j \in C, i \in F_j^\gamma & \\
x_{i,j}, y_i \in \{0, 1\} & \qquad \forall j \in C, i \in F_j^\gamma &
\end{aligned}$$

ILP$^\gamma$ has a variable y_i for each $i \in F$ that indicates whether the 'facility' i is opened, and a variable $x_{i,j}$ for any combination of an original point j and a facility $i \in F_j^\gamma$ that says whether j is connected to i.

Let (x, y) be any solution of ILP$^\gamma$ and let $c(x, y)$ be the cost of the solution. We relate the solutions of ILP$^\gamma$ to the problem (P1) by the following lemmas.

Lemma 3.4.2 *Given a solution (x, y) of ILP$^\gamma$, there exists a solution $\mathcal{C} = \{C_l\}_{l=1}^k$ to* (P1) *which has cost no more than $c_a(\mathcal{C}) \leq (\gamma + 1)c(x, y)$.*

Proof: Since (x, y) is an integral solution, let $\{p_1, \ldots, p_k\} \subseteq F$ denote the facilities which are opened. We define the cluster C_i to be the set of clients j such that $x_{p_i, j} = 1$. For any client

$j \in C_i$, let $j' \in C_i$ be the client which maximizes $D(j, j')$. Since D is a metric, we know that $D(j, j') \leq D(p_i, j) + D(p_i, j')$. By the locality constraint, it holds that $D(p_i, j') \leq \gamma \mathsf{R}_{p_i}$. By definition of D and R_{p_i}, we know $D(p_i, j) \geq \mathsf{R}_{p_i}$, which implies $D(p_i, j') \leq \gamma D(p_i, j)$. Hence, $D(j, j') \leq (\gamma+1)D(p_i, j)$. Summing this over all clients, we conclude that $c_a(\mathcal{C}) \leq (\gamma+1)c(x, y)$. ■

Lemma 3.4.3 *Given a solution $\mathcal{C}$ to* (P1)*, we can construct a solution (x, y) to ILP$^\gamma$(where $\gamma \geq 3$) which has cost $\frac{1}{2}c_a(\mathcal{C}) \leq c(x, y) \leq 2c_a(\mathcal{C})$.*

Proof: Fix a cluster C_i, let $j_1, j_2 \in C_i$ be two clients which maximize $D(j_1, j_2)$. Open facility $p_{j_1 j_2}$ and connect all clients in C_i to it. Notice that it holds $D(j, p_{j_1 j_2}) = \mathsf{R}_{p_{j_1 j_2}} + \min\{D(j, j_1), D(j, j_2)\} \leq 3\mathsf{R}_{p_{j_1 j_2}}$ because $D(j_1, j_2)$ is the maximum distance between two clients in C_i and because $D(j_1, j_2) = 2\mathsf{R}_{p_{j_1 j_2}}$. Thus, the solution is feasible for $\gamma = 3$.

For any client $j \in C_i$, let $j' \in C_i$ be the element which maximizes $D(j, j')$. For the first inequality notice that each client j will pay at least $D(j, p_{j_1 j_2}) \geq \frac{1}{2}D(j_1, j_2) \geq \frac{1}{2}D(j, j')$. Observe that $D(j, j') \geq \max\{D(j, j_1), D(j, j_2)\} \geq (D(j, j_1) + D(j, j_2))/2 \geq D(j_1, j_2)/2$. Combining this with the observation that $D(j, j') \geq \min\{D(j, j_1), D(j, j_2)\}$, we get that

$$D(p_{j_1 j_2}, j) = \min\{D(j, j_1), D(j, j_2)\} + D(j_1, j_2)/2 \leq 2 \cdot D(j, j').$$

Summing over all clients gives the second inequality. ■

Let opt^γ_{ilp} be the optimal value for ILP$^\gamma$ and let opt_a be the value of an optimal solution for (P1). Lemma 3.4.3 implies that $opt^3_{ilp} \leq 2 \cdot opt_a$. Assuming we compute an ψ-approximate solution to the optimal ILP^3 solution that violates the locality constraint by an additional factor of ϱ. Then this solution costs at most $\psi \cdot opt^3_{ilp} \leq 2\psi \cdot opt_a$ and violates the locality constraints by 3ϱ. By Lemma 3.4.2, we can then construct a feasible solution for (P1) that costs at most $(3\varrho+1)\cdot 2\psi \cdot opt_a$.

3.4.3 Good fractional solutions for the local k-median problem

Since problem ILP$^\gamma$ is NP-hard, we relax the integrality constraints to obtain a linear program. The only difference between the standard k-median relaxation and LP$^\gamma_{\mathcal{P}}$ is the locality constraint, i.e., each client j can only connect to facilities in F^γ_j.

$$\begin{aligned}
\min \quad & \sum_{i,j} D(i,j)x_{i,j} && \text{(LP}^\gamma_{\mathcal{P}}\text{)} \\
\text{s.t.} \quad & \sum_{i \in F^\gamma_j} x_{i,j} \geq 1 && \forall j \in C \\
& y_i - x_{i,j} \geq 0 && \forall j \in C, i \in F^\gamma_j \\
& \sum_{i \in F} -y_i \geq -k \\
& x, y \geq 0.
\end{aligned}$$

$$\begin{aligned}
\max \quad & \sum_{j} \alpha_j - kZ && \text{(LP}^\gamma_{\mathcal{D}}\text{)} \\
\text{s.t.} \quad & \\
& \alpha_j \leq D(i,j) + \beta_{i,j} && \forall j \in C, i \in F^\gamma_j \\
& \sum_{j : i \in F_j} \beta_{i,j} \leq Z && \forall i \in F \\
& \alpha, \beta, Z \geq 0.
\end{aligned}$$

The above LP is very similar to the LP for facility location and this fact was exploited by Jain and Vazirani to show that primal-dual solutions to the facility location problem can be transformed into the solutions for the k-median problem. Let LP-F$^{\gamma}_{\mathcal{P}}$ be the facility location variant of LP$^{\gamma}_{\mathcal{P}}$, and let LP-F$^{\gamma}_{\mathcal{D}}$ be its dual:

$$\min \sum_{i\in F, j\in C} D(i,j)x_{i,j} + \sum_{i\in F} f_i y_i \qquad \text{(LP-F}^{\gamma}_{\mathcal{P}}\text{)}$$

$$\text{s.t.} \sum_{i\in F^{\gamma}_j} x_{i,j} \geq 1 \qquad \forall j \in C$$

$$y_i - x_{i,j} \geq 0 \qquad \forall j \in C, i \in F^{\gamma}_j$$

$$x, y \geq 0.$$

$$\max \sum_j \alpha_j \qquad \text{(LP-F}^{\gamma}_{\mathcal{D}}\text{)}$$

$$\text{s.t.}$$

$$\alpha_j \leq D(i,j) + \beta_{i,j} \; \forall j \in C, i \in F^{\gamma}_j$$

$$\sum_{j: i\in F^{\gamma}_j} \beta_{i,j} \leq f_i \qquad \forall i \in F$$

$$\alpha, \beta \geq 0.$$

Augmenting ideas introduced by Jain and Vazirani [93], we obtain integer solutions to LP-F$^{\gamma}_{\mathcal{P}}$. This produces two solutions (x^1, y^1) and (x^2, y^2) that are nearly feasible for LP$^{3\gamma}_{\mathcal{P}}$, but $\sum_i y^1_i = k_1 < k$ and $\sum_i y^2_i = k_2 > k$. A suitable convex combination of the two is a feasible solution for LP$^{3\gamma}_{\mathcal{P}}$ and is a constant factor away from the optimal value of LP$^{\gamma}_{\mathcal{P}}$. Given any $\epsilon > 0$ and $\gamma > 0$, there exists a polynomial time algorithm which finds two feasible integer solutions $(x^1, y^1), (x^2, y^2)$ for LP-F$^{3\gamma}_{\mathcal{P}}$ with the following properties:

1. $\sum_i y^1_i = k_1$ and $\sum_i y^2_i = k_2$ for two integers $k_1 < k < k_2$.
2. Set $\rho = \frac{k_2 - k}{k_2 - k_1}$. The solution $(\hat{x}, \hat{y}) = \rho(x^1, y^1) + (1-\rho)(x^2, y^2)$ is feasible for LP$^{3\gamma}_{\mathcal{P}}$ with cost at most $(3+\epsilon)$ times the optimal solution to LP$^{\gamma}_{\mathcal{P}}$.

Since the essential ideas behind this lemma use standard techniques, we omit the full proof because of space limitations. The main differences to the standard Jain-Vazirani primal-dual process are as follows: When finding the initial set of open facilities, we restrict clients to paying and connecting only to facilities whose radius they lie in. In the clean-up step, the Jain-Vazirani algorithm selects the finally open facilities by finding an arbitrary independent set of facilities in some graph. We use the freedom to choose any independent set and choose a set that ensures that clients that have to be reassigned (because their original facility was closed) can always be routed to an open facility with higher radius than their original facility.

3.4.4 Rounding

Given any two integer solutions (x_1, y_1) and (x_2, y_2) for LP-F$^{\gamma}_{\mathcal{P}}$, which open $A, B \subseteq F$ facilities, respectively, we define a weighted bipartite graph $G(x_1, y_1, x_2, y_2)$ as follows. The graph is defined on the vertex set with bipartitions A and B. We connect $i \in A$ to $i' \in B$ if there exists a client j such that $x^1_{i,j} = 1$ and $x^2_{i',j} = 1$. The weight of an edge (i, i') is the number of clients j which

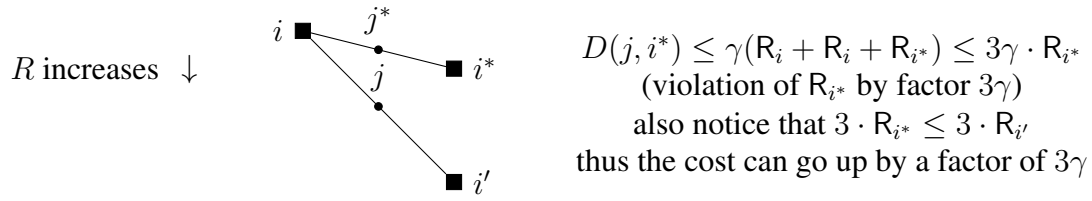

Figure 3.1: Removing all but one down edge for client i.

satisfy the above requirement.

Lemma 3.4.4 *The following holds for local k-median instances that arise from the aversion k-clustering problem. Given two integer solutions $(x^1, y^1), (x^2, y^2)$ for LP-$F^{\gamma}_{\mathcal{P}}$ which open facilities $A, B \subseteq F$, respectively, we can construct solutions $(\tilde{x}^1, \tilde{y}^1), (\tilde{x}^2, \tilde{y}^2)$ that satisfy:*

1. *$(\tilde{x}^1, \tilde{y}^1)$ opens facilities A and $(\tilde{x}^2, \tilde{y}^2)$ opens facilities B.*
2. *If $(x_1, y_1), (x_2, y_2)$ are feasible for LP-$F^{\gamma}_{\mathcal{P}}$, then $(\tilde{x}^1, \tilde{y}^1), (\tilde{x}^2, \tilde{y}^2)$ are feasible solutions to LP-$F^{3\gamma}_{\mathcal{P}}$, and they satisfy $c(\tilde{x}^1, \tilde{y}^1) \le 3\gamma c(x^1, y^1)$ and $c(\tilde{x}^2, \tilde{y}^2) \le 3\gamma c(x^2, y^2)$.*
3. *The graph $G(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ is a forest.*

Proof: We will assume that all radii are distinct (we can ensure this, e.g., by adding a tiny amount of noise to all the radii, or by breaking ties consistently). We say that an edge $\{i, i'\}$ in $G(x^1, y^1, x^2, y^2)$ is a *down edge for i* if $\mathsf{R}_{i'} > \mathsf{R}_i$. For $i \in A \cup B$, let $\mathfrak{D}(i) := \{i' \mid \{i, i'\} \text{ is a down edge}\}$ be the set of facilities that are connected to i by down edges. Furthermore, for every $i \in A \cup B$, let i^* be a facility that minimizes $\{\mathsf{R}_i \mid i \in \mathfrak{D}(i)\}$, i.e., i^* is the endpoint of a 'highest'down edge. For each $i \in A \cup B$, we modify assignments as follows. For all clients $j \in C$ connected to i, and to some facility $i' \in \mathfrak{D}(i)$ in (x_1, y_1), (x_2, y_2), we reassign them to now connect to i and i^* in $(\tilde{x}^1, \tilde{y}^1), (\tilde{x}^2, \tilde{y}^2)$. Thus, for all clients $j \in C$ originally connected to i and i^*, the assignment does not change.

Let us calculate the costs of the resulting assignment. Let $i' \in \mathfrak{D}(i)$ be a facility with $i' \neq i^*$ and let j be a client that is reconnected from i' to i^*. Notice that $D(i, i^*) \le \gamma\mathsf{R}_i + \gamma\mathsf{R}_{i^*}$ since at least one client lies within the (γ-expanded) radius of i and i^* simultaneously. We observe that $D(j, i^*) \le D(j, i) + D(i, i^*) \le \gamma(\mathsf{R}_i + \mathsf{R}_i + \mathsf{R}_{i^*}) \le 3\gamma \cdot \mathsf{R}_{i^*}$ by the triangle inequality and by $\mathsf{R}_{i^*} \ge \mathsf{R}_i$. Thus, the new solution violates the locality constraint for j by a factor of at most 3. Since R_{i^*} is the smallest radius for all facilities in $\mathfrak{D}(i)$, it holds that $\mathsf{R}_{i^*} \le \mathsf{R}_{i'}$. Thus, we also have $D(j, i^*) \le 3\gamma\mathsf{R}_{i'}$. Moreover, since the instances arise from local k-median, equations (3.1) and (3.2) imply that $D(j, i') \ge \mathsf{R}_{i'}$. (This is the only part of the proof that relies on the local k-median instances arising from aversion k-clustering). Hence we have $D(j, i^*) \le 3\gamma\mathsf{R}_{i'} \le 3\gamma D(j, i')$. Thus the cost of each client j is increased by a factor of at most 3γ, which immediately proves Property 2. (Figure 1 visualizes this calculation).

We do not open or close any facilities, thus Property 1 is true. To see Property 3 holds, note that by the distinct radii assumption, any cycle would contain a facility with two down edges, which is no longer possible after the reassignment. ■

Lemma 3.4.4 transforms our solution such that it corresponds to a forest T on the vertices $A \cup B$. We first assume that T is a tree and later deal with each connected component separately.

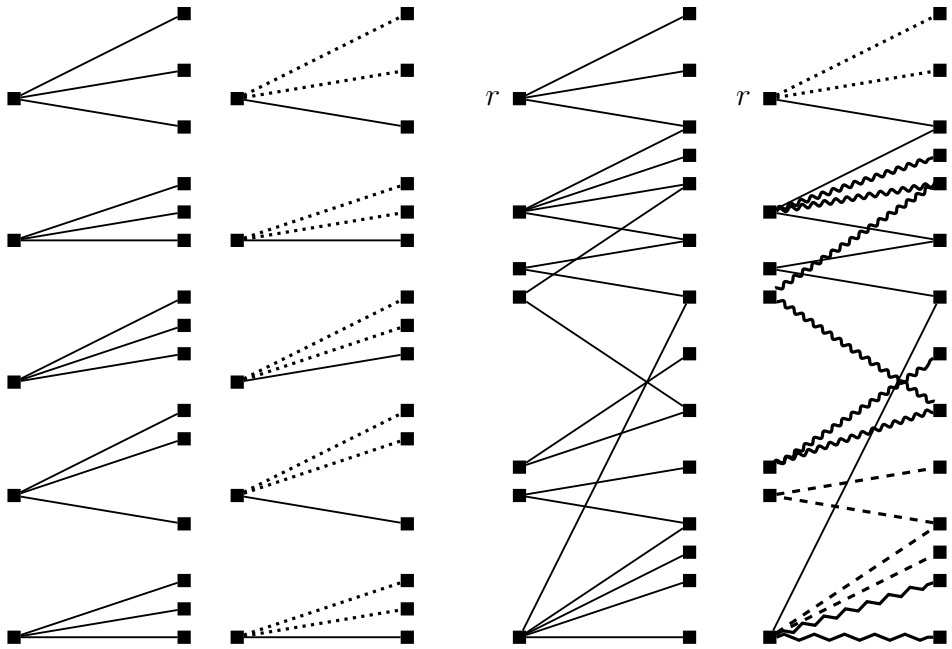

Figure 3.2: Examples on finding subtrees of $G(x^1, y^1, x^2, y^2)$ with $\mathsf{df}(T_i) = 1$. The two left pictures show a simple special case that also is a worst case for the number of subtrees: The deficiency of the shown forest F is $\mathsf{df}(F) = 2k = 10$ and we get $\mathsf{df}(F)/2 = 5$ subtrees by pairing the nodes from B. The two right pictures show a connected tree T with $\mathsf{df}(T) = 7$ and 4 subtrees with $\mathsf{df}(T_i) = 1$.

We use the tree structure to define a depending rounding procedure to combine A and B into an integral solution C with low cost. It will be crucial to look at the difference between the number of vertices from B and A in subtrees of T.

Definition 3.4.5 *For any subtree of $T' \subseteq T$, we define the* deficiency *of T' to be $\mathsf{df}(T') = |B(T')| - |A(T')|$ where $B(T')$ (and $A(T')$) are the vertices from B (and A) in this subtree.*

We start with $C = B$. Then we find a subtree T' with $\mathsf{df}(T') = 1$, i.e., one node more from B than from A. We want to close all facilities in $B(T')$, open all facilities in $A(T')$ and reconnect the affected clients. We want that the reassignment follows the assignments in $(\tilde{x}^1, \tilde{x}^2)$, so all facilities in A that are adjacent to $B(T')$ must be contained in $A(T')$. We then iterate this process with more subtrees until C has exactly k vertices. Since we gain one for every subtree, we need $t := k_2 - k$ subtrees until $|C|$ was reduced from k_2 to k. The subtrees must be disjoint on the B side, while the vertices from A can overlap. The following lemma shows that we can find a large set of suitable subtrees from which we can choose the cheapest t later. Figure 3.2 visualizes two examples.

Lemma 3.4.6 *Given any tree T with $\lceil \mathsf{df}(T)/2 \rceil = l$ and root $r \in A$, we can find l subtrees $T_1, \ldots, T_l$ of T with*

1. $\mathsf{df}(T_i) = 1$
2. $B(T_i) \cap B(T_j) = \emptyset$
3. $A(\delta(B(T_i))) \subseteq A(T_i)$

where we use $\delta(X)$ to denote the set of edges from X to $\bar{X}$.

Proof: Let r be the root of T and $c_1, \ldots, c_\nu$ be the children of r. Our proof will proceed with induction on the height of T. By *removing* a subtree T' we mean that we remove all edges that are in T' from T and all vertices except the root of T'.

Induction Hypothesis: There exist subtrees $T_1, \ldots, T_z$, $z \in \mathbb{N}_0$, of T that satisfy:

1. Each subtree T_i is rooted at a vertex in A and satisfies that $\mathsf{df}(T_i) = 1$.
2. After removing $T_1, \ldots, T_z$ from T, the following holds. If $r \in B$ then $\mathsf{df}(T) \leq 1$. If $r \in A$ then $\mathsf{df}(T) \leq 0$.

Base Case: T has height 0 or 1, i.e., it is a star. If $r \in B$, then $\mathsf{df}(T) \leq 1$ because there is only one node from B. If $r \in A$, then we can remove the children in pairs until there are no pairs left. This is because the subtree consisting of r and any two of its children has deficiency 1. Therefore, each pair and the root will correspond to a subtree (one of T_i mentioned in the IH) that we remove. After removing them, T consists of only r or r and one node from B. In both cases, $\mathsf{df}(T) \leq 0$.

Induction Step: Case $r \in B$: By the induction hypothesis (IH), we can remove some subtrees to ensure that each subtree rooted at $c_1 \ldots c_\nu$ will have deficiency at most 0. Since $\mathsf{df}(T) = \sum_{i=1}^{\nu} \mathsf{df}(T_{\text{rooted at } c_i}) + 1 \leq 1$, we can conclude that this satisfies the first property in the IH. Since we didn't remove any additional subtrees, the second property is vacuously satisfied.
Case $r \in A$: By the IH, we know that the subtree rooted at each child c_i has deficiency $\mathsf{df}(c_i) \leq 1$. Without loss of generality, let $c_1, \ldots, c_p$ be the children which have deficiency 1 and $c_{p+1} \ldots c_\nu$ have deficiency ≤ 0. If $p \leq 1$, then $\mathsf{df}(T) \leq 0$. If $p \geq 2$, then we remove pairs of children with positive deficiency. Observe that the subtree rooted at r containing only the children c_1 and c_2 has deficiency exactly 1. Hence, these satisfy the second property in the IH. We continue this process until there is at most 1 child which has positive deficiency, at which point the the first property is satisfied. This ensures that the induction step is satisfied.

Notice that each removed subtree has deficiency one. Since we keep the root, the deficiency decreases by two for each removed subtree. When $r \in A$ as assumed in the lemma, then $\mathsf{df}(T)$ is decreased to at most zero. Thus, at least $\lceil \mathsf{df}(T)/2 \rceil$ subtrees are removed. ■

For a forest F consisting of trees $F_1, \ldots, F_x$, set $\mathsf{df}(F) := \sum_{j=1}^{x} \mathsf{df}(F_j)$. We can find $\lceil \mathsf{df}(F_j)/2 \rceil$ subtrees satisfying the properties of Lemma 3.4.6 for every every F_j. Thus, we get

$$\sum_{j=1}^{x} \left\lceil \frac{\mathsf{df}(F_i)}{2} \right\rceil \geq \left\lceil \frac{1}{2} \sum_{j=1}^{x} \mathsf{df}(F_i) \right\rceil = \lceil \mathsf{df}(F)/2 \rceil$$

subtrees, giving the following corollary.

Corollary 3.4.7 *Given any forest F with $\lceil \mathsf{df}(F)/2 \rceil = l$, we can find l subtrees $T_1, \ldots, T_l$ of F satisfying the properties from Lemma 3.4.6.*

We now show Theorem 3.3.2. We are given an instance of the local k-median problem that arises from the aversion k-clustering problem. We know that the solutions for the local k-median problem that are induced by the aversion k-clustering instance are feasible for LP-F$^3_{\mathcal{P}}$. Thus, we set $\gamma := 3$. Then we use Lemma 3.4.3 and Lemma 3.4.4 to get two solutions (x^1, y^1) and (x^2, y^2)

so that the graph $G(x^1, y^1, x^2, y^2)$ is a forest, (x^1, y^1) opens k_1 facilities and (x^2, y^2) opens $k_2 \geq k_1$ facilities. Both Lemma 3.4.3 and Lemma 3.4.4 induce a factor of 3 in the radius violation, so (x^1, y^1) and (x^2, y^2) are feasible for LP-F$_{\mathcal{P}}^{9\gamma}$. Furthermore, the intermediate solutions $(\hat{x}^1, \hat{y}^1)$ and $(\hat{x}^2, \hat{y}^2)$ coming from Lemma 3.4.3 have the property that for $\rho = (k_2 - k)/(k_2 - k_1)$, it holds that $\rho \cdot c(\hat{x}^1, \hat{y}^1) + (1 - \rho) \cdot (\hat{x}^2, \hat{y}^2) \leq (3 + \varepsilon) \cdot opt_l^{\gamma}$. Applying Lemma 3.4.4 increases the cost bound by a factor of 3γ. Thus, we know that

$$\rho \cdot c(x^1, y^1) + (1 - \rho) \cdot c(x^2, y^2) \leq (3 + \varepsilon) \cdot 3\gamma \cdot opt_l^{\gamma} := c^{mix}$$

for $\rho = (k_2 - k)/(k_2 - k_1)$. If (x^1, y^1) or (x^2, y^2) opens exactly k facilities, we are done. Otherwise, $k_1 < k < k_2$. If $\rho \geq 1/2$, simply output (x^1, y^1) which then costs $c(x^1, y^1) \leq 2\rho \cdot c(x^1, y^1) \leq 2c^{mix}$. We assume that this is not the case, i.e., $\rho < 1/2$.

We build a solution C and start with $C = B$. Using Corollary 3.4.7, we find $\frac{1}{2}(k_2 - k_1)$ subtrees $T_1, \ldots, T_\ell$ of $G(x^1, y^1, x^2, y^2)$. For each subtree T_i, we can reassign the clients from the facilities in $B(T_i)$ to the facilities in $A(T_i)$. We denote the connection cost for assigning the clients to $A(T_i)$ by $c(T_i)$. Notice that $c(x^1, y^1) \geq \sum_{s=1}^{\ell} c(T_i)$ because every edge of T can only appear in one subtree (since the $B(T_i)$ are pairwise disjoint). Thus, if we choose the $t = k_2 - k$ subtrees $T_{i_1}, \ldots, T_{i_t}$ with the cheapest $c(T_i)$, then

$$\sum_{z=1}^{t} c(T_{i_z}) \leq \frac{t}{\ell} \sum_{s=1}^{\ell} c(T_i) \leq \frac{t}{\ell} c(x^1, y^1) = \frac{k_2 - k}{\frac{1}{2}(k_2 - k_1)} c(x^1, y^1) = 2\rho \cdot c(x^1, y^1).$$

The cost of C starts at $c(x^2, y^2)$ and is increased by at most $2\rho \cdot c(x^1, y^1)$. Thus, the solution costs at most $2\rho \cdot c(x^2, y^2) + c(x^2, y^2) \leq 2\rho \cdot c(x^2, y^2) + 2(1 - \rho) \cdot c(x^2, y^2) \leq 2 \cdot c^{mix}$ where we recall that $\rho < 1/2$. Thus, we get an integer solution of cost $2 \cdot (3 + \varepsilon) \cdot 3\gamma \cdot opt_l^{\gamma}$ that is feasible for LP$_{\mathcal{P}}^{9\gamma}$. That induces a solution for the aversion k-clustering instance that is a constant factor approximation as we described below Lemma 3.4.3.

3.4.5 Improving the Approximation Factor

To improve the final approximation ratio for the aversion k-clustering problem, we observe that the dual variables computed by the primal-dual algorithm can be directly related to the objective of the aversion k-clustering problem. We split each such dual: Let $\alpha_O(j)$ denote the amount that client j pays to open a facility (the subscript O stands for "open"). Using the terminology of Jain and Vazirani, we say a client j is directly connected to facility i if $\beta_{i,j} > 0$ and facility i is open. In this case, $\alpha_O(j) := \beta_{i,j}$. Otherwise, $\alpha_O(j) = 0$. Define $\alpha_C(j) := \alpha(j) - \alpha_O(j)$ (intuitively, this is the connection cost—the subscript C is for connection—that the client has paid for, but for indirectly connected clients we only know that $D(i, j) \geq \alpha_C(j) \geq (1/3)D(i, j)$ is true. For directly connected clients, $\alpha_C(j) = D(i, j)$.).

Lemma 3.4.8 *At the end of the primal-dual algorithm, if client j connects to facility i, then $\alpha_C(j) \geq \mathsf{R}_i$.*

Proof: If j is directly connected to i, then it is immediate that $\alpha_C(j) = D(i, j) \geq \mathsf{R}_i$.

Suppose that j is indirectly connected to facility i'. In this case, let i be the facility that j was first connected to. Since j is indirectly connected, there has to be a client j' that has special edges to both i and i'. We use $t(i)$ and $t(i')$ to denote the times at which facilities i and i' were respectively opened. Notice that $\alpha_C(j) = \alpha(j)$ by definition of α_C for indirectly connected clients and that $\alpha(j) = t(i)$ because j was connected to i before.
Case $t(i) \geq t(i')$: In this case we know that $\alpha_C(j) = t(i) \geq t(i') \geq D(i', j') \geq \mathsf{R}_{i'}$.
Case $t(i) < t(i')$: Since j' has special edges to i and i', it had tight edges to both before either was opened, i.e., $D(i', j') \leq t(i)$. Thus we can say $\alpha_C(j) = t(i) \geq D(i', j') \geq \mathsf{R}_{i'}$. ■

Once again, we may assume that the Jain-Vazirani algorithm returns two solutions (x^1, y^1), (x^2, y^2) and their duals (α^1, Z^+) and (α^2, Z^-). It follows from Jain and Vazirani's analysis that the solutions have the following properties.

1. $\sum_i y_i^1 = k_1$ and $\sum_i y_i^2 = k_2$.
2. $\sum_j \alpha_C^1(j) = \sum_j \alpha_j^1 - k_1 \cdot Z^+$ and $\sum_j \alpha_C^2(j) = \sum_j \alpha_j^2 - k_2 \cdot Z^-$
3. $x_{i,j}^1 = 1$ or $x_{i,j}^2 = 1 \implies D(i, j) \leq 3\gamma \mathsf{R}_i$
4. $|Z^+ - Z^-| \leq \epsilon$
5. For $\rho = \frac{k_2 - k}{k_2 - k_1}$, $\rho(\alpha^1, Z^+) + (1 - \rho)(\alpha^2, Z^-)$ is feasible for $\text{LP}_{\mathcal{D}}^{\gamma}$.

For property 2, notice that $\alpha_C^1(j) = \alpha_j^1$ for indirectly connected clients, that $\alpha^1 C(j) = \alpha_j^1 - \beta_{\phi(j)j}$ for directly connected clients (where $\phi(j)$ is the center j is connected to) and that the sum of $\beta_{\phi(j)j}$ over all directly connected clients is just $k_1 Z^+$. The same holds for the second solution. Using property 5, we get that $\rho(\sum_j \alpha_j^1 - k \cdot Z^+) + (1 - \rho)(\sum_j \alpha_j^2 - k_2 \cdot Z^-)$ is a lower bound for the optimal value of $\text{LP}_{\mathcal{D}}^{\gamma}$ and thus also for the optimal value of $\text{LP}_{\mathcal{P}}^{\gamma}$. Using property 2 and 4, this implies that $\rho(\sum_j \alpha_C^1(j)) + (1-\rho)(\sum_j \alpha_C^2(j)) \leq opt(\text{LP}_{\mathcal{P}}^{\gamma}) + \varepsilon$. We apply Lemma 3.4.4 to replace x^1 and x^2 to ensure that the resulting graph $G(x^1, y^1, x^2, y^2)$ is a forest. Note that the procedure only reassigns the clients to facilities with smaller radius than their currently connected facility. Hence, we can still assume that $x_{i,j}^1 = 1 \implies \alpha_C^1(j) \geq \mathsf{R}_i$ (similarly $x_{i,j}^2 = 1 \implies \alpha_C^2(j) \geq \mathsf{R}_i$). However, we may now have solutions that violate the locality constraints by a factor of 9γ.

Now we use the procedure described in Lemma 3.4.6 to partition the graph $G(x^1, y^1, x^2, y^2)$ into subtrees $T_1, \ldots T_\ell$ with $\mathsf{df}(T_p) = 1$ for $p \in \{1, \ldots, \ell\}$ and $\ell = \frac{k_2 - k_1}{2}$. Each tree has the property that $A(\delta(B(T_p))) \subseteq A(T_p)$ for all $p \in \{1, \ldots, \ell\}$. Since each edge in this tree represents some set of clients, we use the notation $j \in T_p$ to denote that j is associated with an edge in T_p. Define the cost of the subtree T_p as $\sum_{j \in T_p} \alpha_C^1(j)$. We choose the $k_2 - k$ cheapest such trees. Since choosing all ℓ subtrees will result in a cost of $\sum_j \alpha_C^1(j)$, we can say that the cost of these chosen subtrees is at most $\frac{2(k_2-k)}{k_2-k_1} \sum_j \alpha_C^1(j) = 2\rho \sum_j \alpha_C^1(j)$.

Our rounded solution $(\hat{x}, \hat{y})$ opens all facilities from A that are part of the chosen subtrees and all facilities from B that are not part of any chosen subtree. Notice that since we open $k_2 - k$ subtrees and these satisfy $\mathsf{df}(T_p) = 1$, $\hat{y}$ opens exactly k facilities. The assignments of clients to facilities follow x^1 and x^2, respectively.

Analogously to Lemma 3.4.2, we construct a solution to the aversion k-clustering problem based on $\hat{x}, \hat{y}$. In this solution, each client assigned to a facility $p_i \in A$ pays at most

$$D(j, j') \leq D(p_i, j) + D(p_j, j') \leq 9\gamma \mathsf{R}_i + 9\gamma \mathsf{R}_i \leq (2 \cdot 9\gamma) \alpha_C^1(j)$$

where j' is the furthest away client among all that are assigned to p_i. Thus, by our choice of subtrees, all clients assigned to A pay at most $(2 \cdot 9\gamma)2\rho \sum_j \alpha_C^1(j)$ in total. The remaining clients pay at most $(2 \cdot 9\gamma)(\sum_j \alpha_C^2(j))$. As before, we can assume that $\rho \leq 1/2$. We conclude that the cost of $(\hat{x}, \hat{y})$ is bounded by

$$\begin{aligned}
&(2 \cdot 9\gamma)\big(2\rho \sum_j \alpha_C^1(j) + \sum_j \alpha_C^2(j)\big) \\
&\leq 2(2 \cdot 9\gamma)(1+\varepsilon)\big(\rho \sum_j \alpha_C^1(j) + (1-\rho)\sum_j \alpha_C^2(j)\big) \\
&\leq 2(2 \cdot 9\gamma)(1+\varepsilon) opt(\text{LP}_{\mathcal{P}}^{\gamma}) \\
&\leq 2(2 \cdot 9\gamma)2(1+\varepsilon) opt_a
\end{aligned}$$

where opt_a is the optimal value for the aversion k-clustering instance and the last inequality follows by Lemma 3.4.3. Since $\gamma = 3$, the approximation factor is bounded by $216 + \varepsilon$.

3.5 Open Problems

Our proof of Theorem 3.3.2 only uses the fact that instance arises from aversion k-clustering in one instance. This raises the natural question:

Problem 3.5.1 *Does there exist a $(O(1), O(1))$-bicriteria algorithm for the Local k-median problem?*

A second question is whether these techniques can be extended to the Balanced k-median problem. There is a natural linear programming based relaxation for this problem, which was first mentioned by Bartal et al. [22]. However, it is not known how to round this linear program even in extremely special cases. We state this is as another direction of possible work.

Problem 3.5.2 *Does the standard linear programming relaxation for the Balanced k-median problem have a $O(1)$ integrality gap?*

This problem is interesting even for special metrics; when all the facilities are located at a single point, the integrality gap of the standard linear program is not known. Note that there is a simple dynamic program that achieves this ratio. We rule out several standard approaches to round this problem. Firstly, we can construct examples where the naive local search achieves an approximation ratio of $\Omega(n)$. Primal-dual techniques similar to the work of Jain and Vazirani [93] and Chuzhoy and Rabani [40] fail because the linear program is not defined over a metric space. In particular, the distance of a facility to a client depends on the capacity of facility and this is a major bottleneck in obtaining a Lagrange multiplier preserving algorithm.

Chapter 4

Fractionally Sub-additive Network Design

4.1 Introduction

We study a robust version of a single-sink network design problem that we call the *Single-sink fractionally-subadditive network design* (f-SAND) problem. In an instance of f-SAND, we are given an undirected graph $G = (V, E)$ with edge costs $w_e \geq 0$ for all $e \in E$, a root node $r \in V$, and k *colors* represented as vertex subsets $C_i \subseteq V \setminus \{r\}$ for all $i \in [k]$, that wish to send flow to r. A feasible solution is an integer capacity installation on the edges of G, such that for every $i \in [k]$, each node in C_i can *simultaneously* send one unit of flow to r. Thus, the total flow sent by color i nodes is is $|C_i|$ while the flows sent from nodes of different colors are instead *non-simultaneous* and can share capacity. An optimal solution is a feasible one that minimizes the total cost of the installation.

Problem 4.1.1 *Given an undirected graph $G = (V, E)$ with edge costs $w_e \geq 0$, a root node $r \in V$, and k* colors *represented as vertex subsets $C_i \subseteq V \setminus \{r\}$ for all $i \in [k]$, that wish to send flow to r. Find the minimum weight set of edges that we must install to ensure that each color C_i can* simultaneously *send one unit of flow to r.*

The single-sink nature of the problem suggests a natural *cut*-covering formulation, namely:

$$\begin{aligned} \min \sum_{e \in E} w_e x_e \quad & \text{s.t.} \\ \sum_{e \in \delta(S)} x_e \geq f(S) \qquad & \forall S \subset V \setminus \{r\} \end{aligned} \tag{IP}$$

where $\delta(S)$ denotes the set of edges with exactly one endpoint in S, and

$$f(S) := \max_{i \in [k]} \{|C_i \cap S|\} \tag{4.1}$$

for all $S \subseteq V \setminus \{r\}$. Despite having exponentially many constraints, the LP-relaxation of (IP) can be solved in polynomial-time because the separation problem reduces to performing k max-flow computations. The main challenge is to round the resulting solution into an integer solution.

Rounding algorithms for the LP relaxation of (IP) have been investigated by many authors, under certain assumptions of the function $f(S)$. Prominent examples are some classes of 0/1-functions (such as *uncrossable* functions), or integer-valued functions such as *proper* functions, or *weakly supermodular* functions [70, 92]; however, these papers consider arbitrary cut requirements rather than the single-sink connectivity requirements we study.

Our single-sink problem is a special case of a broader class of subadditive network design problems where the function f is allowed to be a general subadditive function. Despite their generality, the single-sink network design problem for general subadditive functions can be approximated within an $O(\log |V|)$ factor by using a tree drawn from the probabilistic tree decomposition of the metric induced by G using the results of Fakcharoenphol, Rao, and Talwar [56], and installing the required capacity on the tree edges. Hence, a natural direction is to consider special cases of such subadditive cut requirement functions.

Our function $f(S)$ defined in (4.1) is an interesting and important special case of subadditive functions. It was introduced as XOS-functions (max-of-sum functions) in the context of combinatorial auctions by Lehman et al. [115]. Feige [59] proved that this function is equivalent to fractionally-subadditive functions which are a strict generalization of submodular functions (hence the title). These functions have been extensively studied in the context of learning theory and algorithmic game theory [14, 26, 115]. Our work is an attempt to understand their behavior as single-sink network design requirement functions.

f-SAND was first studied by Oriolo et al. [125] in the context of *robust* network design, where the goal is to install minimum cost capacity on a network in order to satisfy a given set of (non-simultaneous) traffic demands among terminal nodes. Each subset C_i can in fact be seen as a way to specify a distinct traffic demand that the network would like to support. They observed that f-SAND generalizes the Steiner tree problem: an instance of the Steiner tree problem with $k+1$ terminals $t_1, \ldots, t_{k+1}$ is equivalent to the f-SAND instance with $r := t_{k+1}$ and $C_i := \{t_i\}$ for all $i \in [k]$. This immediately shows that f-SAND is NP-Hard (in fact, APX-hard [39]) when k is part of the input. The authors in [125] strengthened the hardness result by proving that f-SAND is NP-Hard even if k is not part of the input, and in particular for $k = 3$ (if $k = 1$ the problem is trivially solvable in polynomial-time by computing a shortest path tree rooted at r). From the positive side, they observed that there is a trivial k-approximation algorithm, that relies on routing via shortest paths, and an $O(\log |\cup_i C_i|)$-approximation algorithm using metric embeddings [56, 77]. The authors conclude their paper mentioning two open questions, namely whether the problem is polynomial-time solvable for $k = 2$, and whether there exists an $O(1)$-approximation algorithm.

4.2 Related works

Network design problems where the goal is to build a minimum cost network in order to support a given set of flow demands, have been extensively studied in the literature (we refer to the survey [36]). There has been a huge amount of research focusing on the case the set of demands is described via a polyhedron (see e.g. [24]). In this context a very popular model is the *Virtual private network* [48, 63], for which many approximation results have been developed (see e.g. [73, 74, 76] and the references therein). For the case where the set of demands is instead given as a (finite)

discrete list, the authors in [125] developed a constant factor approximation algorithm on ring networks, and proved that f-SAND is polynomial-time solvable on ring networks.

Regarding the formulation (IP), Goemans and Williamson [70] gave a $O(\log(f_{max}))$-approximation algorithm for solving (IP) whenever $f(S)$ is an integer-valued proper function that can take values up to f_{max}, based on a primal-dual approach. Subsequently, Jain [92] improved this result by giving a 2-approximation algorithm using iterative rounding of the LP-relaxation. Recently, a strongly-polynomial time FPTAS to solve the LP-relaxation of (IP) with proper functions has been given in [62].

4.3 Our results

In this chapter, we answer the first open question in [125] by showing that f-SAND is NP-Hard for $k = 2$ via a reduction from SAT. We give a $\frac{3}{2}$-approximation algorithm for this case ($k = 2$). This is the first improvement over the (trivial) k-approximation obtained using shortest paths for any k. Our approximation algorithm is based on pairing terminals of different groups together, and therefore reducing to a suitable minimum cost matching problem. While the idea behind the algorithm is natural, its analysis requires a deeper understanding of the structure of the optimal solution.

We also introduce an interesting variant of f-SAND, which we call the Latency-f-SAND problem, where the network built is restricted to being a *path* with the root r being one of the endpoints (f-SAND-path). We show a $O(\log^2 k \log n)$-approximation using a new reformulation of the problem that allows us to exploit techniques recently developed for *latency* problems [31].

While being a generalization of well-studied problems, f-SAND does not seem to admit an easy $O(1)$-approximation via standard LP-rounding techniques for arbitrary values of k. We prove some structural results that highlight the difficulty of the general problem. In particular, we show a family of a instances providing a super-constant gap between an optimal f-SAND solution and an optimal *tree*-solution, i.e., a solution whose support is a tree – this rules out many methods that output a solution with a tree structure. The bulk of the construction was shown in [72] and we amend it to our problem using a simple observation. Furthermore, we give some evidence that an iterative rounding approach (as in Jain's fundamental work [92]) is unlikely to work. This follows by considering a special class of Kneser Graphs, where the LP seems to put low fractional weight on each edge in an extreme point.

4.4 $3/2$-approximation for the two color case

The goal of this section is to give a $\frac{3}{2}$-approximation algorithm for SAND with two colors. We remark that our algorithm bypasses the difficulties mentioned in the previous section. In particular, the final output is not a tree.

4.4.1 Simplifying Assumptions.

We will refer to the two colors as *green* and *blue*, and let $C_G \subset V$ denote the set of green terminals, and $C_B \subset V$ denote the set of blue terminals. Without loss of generality, we will assume that $|C_G| = |C_B|$, i.e., the cardinality of green terminals is equal to the cardinality of blue terminals (if not, we could easily add dummy nodes at distance 0 from the root). Furthermore, by replacing each edge in the original graph with $|C_G|$ parallel edges of the same cost, we can assume that in a feasible solution the capacity installed on each edge must be either 0 or 1. This means that each edge is used by *at most* one terminal of C_G (resp. C_B) to carry flow to the root. Lastly, we assume that every terminal in C_G shares at least one edge with some terminal in C_B in the optimal solution.[1]

Let OPT denote an optimal solution to a given instance of SAND with two colors. We start by developing some results on the structure of OPT, that will be crucial to analyze our approximation algorithm later.

4.4.2 Understanding the structure of OPT

A feasible solution of a SAND instance consists of a (integer valued) capacity installation on the edges that allows for a flow from the terminals to the root. Given a feasible solution, each terminal will send its unit of flow to t on a single path. Let us call the collection of such paths a *routing* associated with the feasible solution. The first important concept we need is the concept of splits.

Shared Edges and Splits.

Given a routing, for each terminal $g \in C_G$ (and $b \in C_B$ respectively) let P_g (P_b) denote the path along which g (b) sends flow to the root; i.e. $P_g := \{g = x_0, x_1, \ldots, x_{|P_g|} = r\}$. We say that an edge e is **shared** if the paths of two terminals of different color contain the edge. We say that $g \in C_G$ and $b \in C_B$ are **partners** with respect to a shared edge $e = uv$, if their respective paths use the edge e; i.e. $e \in P_g \cap P_b$.

Definition 4.4.1 *A **split** in the path P_g is a maximal set of consecutive edges of the path such that g is partnered with some b on all the edges of this set.*

If $\{(x_i, x_{i+1}), (x_{i+1}, x_{i+2}), \ldots, (x_{i+j-1}, x_{i+j})\}$ is a split in the path $P_g = \{g = x_0, x_1, \ldots, x_{|P_g|} = r\}$ for $g \in C_G$, then there exists a unique terminal $b \in C_B$ such that P_b contains the edges $\{(x_i, x_{i+1}), (x_{i+1}, x_{i+2}), \ldots, (x_{i+j-1}, x_{i+j})\}$, P_b does not contain the edge (x_{i-1}, x_i), and if $x_{i+j} \neq r$ then P_b does not contain the edge (x_{i+j}, x_{i+j+1}). By our assumptions, the terminal b is unique as each edge is used by at most one terminal of each color.

Since the flow is going from a terminal g to r, the path P_g naturally induces an orientation on its edges given by the direction of the flow, even though the edges are undirected. Of course, the paths of different terminals could potentially induce opposite orientations on (some of) the shared edges (see Figure 4.1).

[1] We can easily ensure this e.g. by modifying our instance as follows: we add a dummy node r' which is only connected to r with $|C_G|$ parallel edges of 0 cost, and we make r' be the new root. In this way, all terminals will use one copy of the edge (r, r').

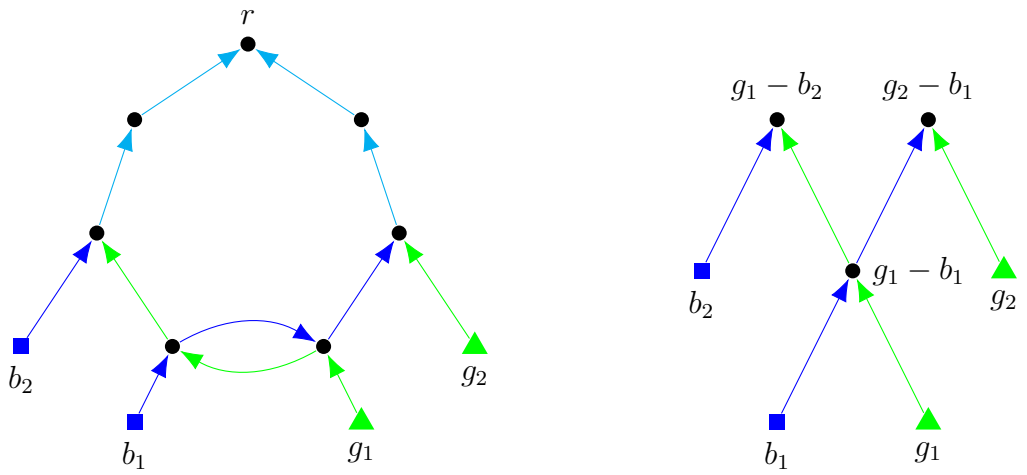

Figure 4.1: The above left graph (where each undirected edge is supposed to have unit capacity) shows an optimal routing. Both b_1 and g_2 (resp. b_2 and g_1) send flow to r going counterclockwise (resp. clockwise) on the edges of the cycle. The path P_{b_1} contains two splits: the first is wide (b_1 is partnered with g_1), the second is thin (b_1 is partnered with g_2). The graph on the right is the Split Graph for the optimal solution on the left. The pair of vertices g_1, b_1 and the pair of vertices g_2, b_2 constitute the fresh pairs.

Definition 4.4.2 *A split is **wide**, if the paths of the two terminals that are partners on the edges of the split induce opposite orientations on the edges. A split is **thin**, if the paths of the two terminals that are partners on the edges of the split induce the same orientation on the edges.*

The above notions are well defined for any routing with respect to a feasible solution. Now, we focus on the structure of an optimal routing, i.e., a routing with respect to an optimal solution. For the rest of this section, we let $\{P_g\}_{g \in C_G}$ and $\{P_b\}_{b \in C_B}$ be an optimal routing. The following lemma is immediate.

Lemma 4.4.3 *Let $\{(x_i, x_{i+1}), (x_{i+1}, x_{i+2}), \ldots, (x_{i+j-1}, x_{i+j})\}$ be a split in the path P_g (for some $g \in C_G$). The edges of the split form a shortest path from x_i to x_{i+j}.*

Proof: If not, we could replace this set of edges with the set of edges of a shortest path from x_i to x_{i+j}, in both P_g and P_b, where b is the partner of g on the split. Therefore, we can install one unit of capacity on these edges, and remove the unit of capacity from the edges of the split. We get another feasible solution with smaller cost, a contradiction to the optimality of our initial solution. ■

Split Graph.

A consequence of Lemma 4.4.3 is each split is entirely characterized by the endpoints of the split and the terminals that share them. We denote each split by a tuple (u, v, g, b) which states that there is a shortest path between u and v whose edges are shared by g and b.

Let $\S$ denote the set of all splits in the optimal routing. We construct a directed graph $G^{\S}$ whose vertex set corresponds to $V = \S \cup C_G \cup C_B$ (i.e. the vertex set contains one vertex for

each split and one vertex for each terminal). For each $g \in C_G$, we place a directed *green* edge going between two consecutive splits in P_g. Specifically, if $\{(x_i, x_{i+1}), \ldots, (x_{i+j-1}, x_{i+j})\}$ and $\{(x_{i'}, x_{i'+1}), \ldots, (x_{i'+j'-1}, x_{i'+j'})\}$ are two splits in P_g with $i < i'$, we say that they are consecutive if the subpath from x_{i+j} to $x_{i'}$ does not contain any split. In this case, we place a directed edge in $G^{\S}$ whose tail is the vertex corresponding to the first split, and whose head is the vertex corresponding to the second one. Similarly, for each $b \in C_B$ we place a directed *blue* edge between vertices of consecutive splits that appear in P_b. Furthermore, for each $g \in C_G$ (resp. $b \in C_B$) we place a directed green (resp. blue) edge from g (resp. b) to the vertex corresponding to the first split on the path P_g (resp. P_b), if any. This graph is denoted as the **Split Graph** (see Figure 4.1).

Each split indicates that two terminals of different colors are sharing the capacity on a set of edges in an optimal routing. Hence, each split-vertex in $G^{\S}$ has indegree 2 (in particular, one edge of each color). Furthermore, each split-vertex in $G^{\S}$ has outdegree either 0 or 2; if it has two outgoing edges, one is green and one is blue. Similarly, each terminal has indegree 0, and outdegree 1 (as we assume that each terminal shares at least one edge).

Fresh Pairs.

We need one additional definition before proceeding to the algorithm.

Definition 4.4.4 *An §-**alternating sequence** is a sequence of vertices of the Split Graph $\{v, s_1, s_2, \ldots, s_h, w\}$ with $h \geq 1$, that satisfies the following:*

(i) (v, s_1) and (w, s_h) are directed edges in $G^{\S}$ and v, w are terminals of different color.

(ii) For all even $i \geq 2$, (s_i, s_{i-1}) and (s_i, s_{i+1}) are both directed edges in $G^{\S}$ with opposite colors.

*We call the path obtained by taking the edges in (i) and (ii) an §-**alternating path**. We call (v, w) a **fresh pair** if they are the endpoints of an §-alternating path.*

By definition, in an §-alternating sequence the vertices $s_1, \ldots, s_h$ are all split-vertices, and h is odd. We remark here that an §-alternating path is *not* a directed path. (See again Figure 4.1).

Lemma 4.4.5 *We can find a set of edge-disjoint §-alternating paths in the Split Graph such that each terminal is the endpoint of exactly one path in this set.*

Proof: We construct the desired set as follows. For each vertex $g \in C_G$, there is a unique outgoing edge to a split vertex $s \in \S$ (as we assume every terminal participates in a split). Since each split-vertex has indegree 2, s has another ingoing edge coming from a different vertex w. If $w \in C_B$, then (v, w) is a fresh pair and we have found an §-alternating sequence $\{v, s, w\}$. If w is a split-vertex, then it has another outgoing edge to a different split-vertex s', which in its turn has another incoming edge from a different vertex w'. We continue to build an alternating sequence (and a corresponding alternating path) in this way until it terminates in a terminal. Since the path is of even length and the colors alternate, we can conclude that this will terminate in a terminal of opposite color. We remove the edges of this path from the Split Graph, and iterate the process. Each terminal will belong to exactly one §-alternating path, as it has outdegree exactly 1, and all the paths are edge-disjoint, proving the lemma. ■

4.4.3 The Algorithm

We are now ready to present our *matching algorithm*. The algorithm has two steps. First, construct a complete bipartite graph $\mathcal{H}$ with the bipartitions C_G and C_B, where the weight on the edge $(g, b) \in C_G \times C_B$ is equal to the cost of the Steiner tree in G connecting g, b and the root. Note that the graph $\mathcal{H}$ can be computed in polynomial time, since a Steiner tree on 3 vertices can be easily computed in polynomial time.

Second, find a minimum-weight perfect matching $\mathcal{M}$ in $\mathcal{H}$, and for each edge $(g, b) \in \mathcal{M}$ install (cumulatively) one unit of capacity on each edge of G that is in the Steiner tree associated to the edge $(g, b) \in \mathcal{M}$. The capacity installation output by this procedure is a feasible solution to f-SAND, and has total cost equal to the weight of $\mathcal{M}$.

Lemma 4.4.6 *The matching algorithm is a $\frac{3}{2}$-approximation algorithm.*

Proof: First, we partition OPT into four parts; let w_b (and w_g respectively) be the cost of the edges which are used only by blue (green respectively) terminals in OPT, and let w_t (w_d) be the cost of edges in thin (wide) splits in OPT. Thus, $w(OPT) = w_b + w_g + w_t + w_d$. By Lemma 4.4.5, we can extract from the Split Graph associated to OPT a set of §-alternating paths such that each terminal is contained in exactly one fresh pair. Consider the matching $\mathcal{M}_1$ determined by the set of fresh pairs found by the aforementioned procedure. We will now bound the weight of $\mathcal{M}_1$.

Claim 4.4.7 *The weight of the matching formed by connecting the fresh pairs is at most*

$$\frac{3}{2} \cdot w_b + \frac{3}{2} \cdot w_g + 1 \cdot w_t + 3 \cdot w_d.$$

Proof: Let (g, b) be a fresh pair and $(g, s_1, \ldots, s_h, b)$ be the corresponding §-alternating sequence. The edges of the associated §-alternating path naturally correspond to paths in G composed by non-shared edges (that connect either the endpoints of two different splits, or one terminal and one endpoint of a split). These paths together with the edges of the wide splits in the sequence, naturally yield a path $P(b, g)$ in G connecting g and b.

If we do this for all fresh pairs, we obtain that the total cost of the paths $P(b, g)$ is upper bounded by $1 \cdot w_b + 1 \cdot w_g + 2 \cdot w_d$. The reason for having a coefficient of 2 in front of w_d is because the §-alternating paths of Lemma 4.4.5 are edge-disjoint, but not necessarily vertex-disjoint: however, since each split-vertex has at most 4 edges incident into it, it can be part of at most 2 §-alternating paths.

Using the aforementioned connection, we can move all terminals in C_G to their partners in C_B. Subsequently, we connect them to the root using the P_b for all $b \in C_B$. This connection to the root will incur a cost of $1 \cdot w_b + 1 \cdot w_d + 1 \cdot w_t$. Combining this together, we get a total cost of $2 \cdot w_b + 1 \cdot w_g + 1 \cdot w_t + 3 \cdot w_d$. Analogously, if we connect the partners in C_G to the root using the the path P_g for all $g \in C_G$, we will incur a total cost of $1 \cdot w_b + 2 \cdot w_g + 1 \cdot w_t + 3 \cdot w_d$. Since the sum of the cost of the Steiner trees connecting the fresh pairs to the root is no more than either of these two values, we can bound the weight of $\mathcal{M}_1$ by their average:

$$\frac{3}{2} \cdot w_b + \frac{3}{2} \cdot w_g + 1 \cdot w_t + 3 \cdot w_d.$$

■

Claim 4.4.8 *There exists a matching in $\mathcal{H}$ of weight at most $1 \cdot w_b + 1 \cdot w_g + 2 \cdot w_t$.*

Proof: Consider the flow routed on the optimal paths by the set of all terminals $C_G \cup C_B$. We modify the flow (and the corresponding routing) as follows. Whenever two terminals traverse a wide-split, re-route the flows so as to not use the wide-split. This is always possible as the two terminals traverse these edges in opposite directions (by definition of wide splits). This re-routing ensures that all the edges of wide-splits are not used anymore in the resulting paths. However, thin-splits which contained terminals of different colors passing in the same direction, might now contain two terminals of the same color passing through the edges. This means that these edges will be used twice (or must have twice the capacity installed). All other edges do not need to have their capacity changed. Thus, the resulting flow can be associated with a feasible solution of cost at most $1 \cdot w_b + 1 \cdot w_g + 2 \cdot w_t + 0 \cdot w_d$. This flow corresponds to all vertices directly connecting to the root as any shared edge is counted twice. Hence, this is a bound on any matching in $\mathcal{H}$. ■
The average weight of the above matchings is an upper-bound on the minimum weight of a matching in $\mathcal{H}$. Hence, the weight of $\mathcal{M}$ is at most

$$\begin{aligned}&\frac{1}{2} \cdot \left(\frac{3}{2} \cdot w_b + \frac{3}{2} \cdot w_g + 1 \cdot w_t + 3 \cdot w_d\right) + \frac{1}{2} \cdot \left(1 \cdot w_b + 1 \cdot w_g + 2 \cdot w_t + 0 \cdot w_d\right) \\ &\leq \frac{3}{2} \cdot \left(w_b + w_g + w_t + w_d\right)\end{aligned}$$

Therefore, the matching algorithm is $\frac{3}{2}$-approximation algorithm. ■

4.5 Hardness for two colors

We prove that the SAND problem is NP-hard even with just two colors.

Theorem 4.5.1 *The SAND problem with 2 colors is NP-hard.*

Proof: We use a reduction from a variant of the Satisfiability (SAT) problem, where each variable can appear in at most 3 clauses, that is known to be NP-hard [141]. Formally, in a SAT instance we are given m clauses $K_1, \ldots, K_m$, and p variables $x_1, \ldots, x_p$. Each clause K_j is a disjunction of some *literals*, where a literal is either a variable x_i or its negation $\bar{x}_i$, for some i in $1, \ldots, p$. The goal is to find a truth assignment for the variables that satisfies all clauses, where a clause is satisfied if at least one of its literals takes value *true*. In the instances under consideration, each variable x_i appears in at most 3 clauses, either as a literal x_i, or as a literal $\bar{x}_i$. It is not difficult to see that, without loss of generality, we can assume that every variable appears in exactly 3 clauses. Furthermore, by possibly replacing all occurrences of x_j with $\bar{x}_j$ and vice versa, we can assume that each variable x_i appears in exactly one clause in its negated form ($\bar{x}_i$).

Given such a SAT instance, we define an instance of SAND as follows (see Fig. 4.2). We construct a graph $G = (V, E)$ by introducing one sink node r, one node k_j for each clause K_j, and 7 distinct nodes y_i^ℓ, ($\ell = 1, \ldots, 7$), for each variable x_i. That is,

$$V := \{r\} \cup \{k_1, \ldots, k_m\} \cup \left\{ \bigcup_{i=1}^{p} \{y_i^1, y_i^2, y_i^3, y_i^4, y_i^5, y_i^6, y_i^7\} \right\}$$

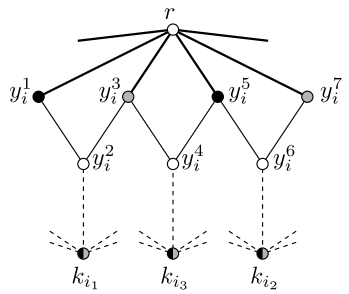

Figure 4.2: The picture shows the subgraph introduced for every variable x_i. Bold edges have cost 2, solid edges have cost 1, and dashed edges have cost M. Black circles indicate nodes in C_1, and grey circles indicate nodes in C_2. Nodes in $C_1 \cap C_2$ are colored half-black and half-grey.

The set of edges E is the disjoint union of three different sets, $E := E_1 \cup E_2 \cup E_3$, where:

$$E_1 := \bigcup_{i=1}^{p} \left\{ \bigcup_{\ell=1}^{4} \{r, y_i^{2\ell-1}\} \right\}; \quad E_2 := \bigcup_{i=1}^{p} \left\{ \bigcup_{\ell=1}^{6} \{y_i^{\ell}, y_i^{\ell+1}\} \right\}.$$

To define the set E_3, we need to introduce some more notation. For a variable x_i, we let i_1 and i_2 be the two indices of the clauses containing the literal x_i, and we let i_3 be the index of the clause containing the literal $\bar{x}_i$. We then have

$$E_3 := \bigcup_{i=1}^{p} \left\{ \{y_i^2, k_{i_1}\}, \{y_i^4, k_{i_3}\}, \{y_i^6, k_{i_2}\} \right\}.$$

We assign cost 2 to the edges in E_1, unit cost to the edges in E_2, and a big cost $M >> 0$ to the edges of E_3 (in particular, $M > 2m + 8p$). Finally, we let the color classes[2] be defined as:

$$C_1 := \{k_1, \ldots, k_m\} \cup \left\{ \bigcup_{i=1}^{p} \{y_i^1, y_i^5\} \right\}; \quad C_2 := \{k_1, \ldots, k_m\} \cup \left\{ \bigcup_{i=1}^{p} \{y_i^3, y_i^7\} \right\}.$$

We claim that there exists an optimal solution to the SAND instance of cost at most $(M + 2)m + 8p$ if and only if there is a truth assignment satisfying all clauses for the SAT instance.

4.5.1 Completeness

First, let us assume that the SAT instance is satisfiable. For each clause K_j, we select one literal that is set to *true* in the truth assignment. We define the paths for our terminal nodes in C_1 as follows. For each node $y \in \bigcup_{i=1}^{p} \{y_i^1, y_i^5\}$, we let the flow travel from y to r along the edge $\{y, r\}$. For each k_j, we let the flow travel to r on a path P_1^j, that we define based on the literal selected for K_j. Specifically, let x_i be the variable corresponding to the literal selected for the clause K_j. Then:

[2]We here have $C_1 \cap C_2 \neq \emptyset$. However, the reduction can be easily modified to prove hardness of instances where $C_1 \cap C_2 = \emptyset$, by simply adding for all j two nodes k_j^1, k_j^2 adjacent to k_j with an edge of zero cost, and by letting k_j^1 (resp. k_j^2) be in C_1 (resp. C_2) instead of k_j.

- if $K_j = K_{i_1}$, we let P_1^j be the path with nodes $\{k_j, y_i^2, y_i^3, r\}$,
- if $K_j = K_{i_2}$, we let P_1^j be the path with nodes $\{k_j, y_i^6, y_i^7, r\}$,
- if $K_j = K_{i_3}$, we let P_1^j be the path with nodes $\{k_j, y_i^4, y_i^3, r\}$.

We define the paths for our terminal nodes in C_2 similarly. For each node $y \in \bigcup_{i=1}^{p}\{y_i^3, y_i^7\}$, we let the flow travel from y to r along the edge $\{y, r\}$. For each k_j, we let the flow travel to r on a path P_2^j defined as follows. Let x_i be the variable corresponding to the literal selected for the clause K_j. Then:

- if $K_j = K_{i_1}$, we let P_2^j be the path with nodes $\{k_j, y_i^2, y_i^1, r\}$,
- if $K_j = K_{i_2}$, we let P_2^j be the path with nodes $\{k_j, y_i^6, y_i^5, r\}$,
- if $K_j = K_{i_3}$, we let P_2^j be the path with nodes $\{k_j, y_i^4, y_i^5, r\}$.

Note that the paths of terminals belonging to the same color set do not share edges. In fact, by construction, the paths of two terminals in C_1 could possibly share an edge only if for two distinct clauses $K_j \neq K_{j'}$ we selected a literal corresponding to the same variable x_i, and we have $K_j = K_{i_1}$ and $K_{j'} = K_{i_3}$, since in this case the paths P_1^j and $P_1^{j'}$ would share the edge $\{y_i^3, r\}$. However, selecting x_i for K_{i_1} means x_i takes value *true* in the truth assignment, while selecting x_i for K_{i_3} means x_i takes value *false* in the truth assignment, which is clearly a contradiction. A similar observation applies to paths of terminals in C_2. It follows that installing one unit of capacity on every edge that appears in (at least) one selected path is enough to support the flow of both color sets. The total installation cost is exactly $8p + (M + 2)m$.

4.5.2 Soundness

Suppose there is an optimal solution to the SAND instance of cost at most $(M + 2)m + 8p$. Let $\mathcal{S}$ denote such solution. Since the support of any feasible solution has to include at least one distinct edge of cost M for each node k_j, and $M > 2m + 8p$, it follows that $\mathcal{S}$ has exactly m edges of cost M in its support, each with one unit of capacity installed. Hence, if we denote by P_1^j (resp. P_2^j) the path used by k_j to send flow to r with terminals in C_1 (resp. C_2), we have the following fact.

Fact 1. For each $j = 1, \ldots, m$, the paths P_1^j and P_2^j from k_j to r share the first edge.

We use this insight to construct a truth assignment for the SAT variables. Specifically, let y_i^ℓ be the endpoint of the first edge of P_1^j and P_2^j. We set x_i to *true* if $y_i^\ell = y_i^2$ or if $y_i^\ell = y_i^6$, and we set x_i to *false* if $y_i^\ell = y_i^4$. We repeat this for all clauses $j = 1, \ldots, m$, and we assign an arbitrary truth value to all remaining variables, if any. In order to finish the proof, we have to show that this assignment is consistent for all $i = 1, \ldots, p$. To this aim, let us say that a variable x_i is *in conflict* if there is a node k_j sending flow to r on a path whose first edge has endpoint y_i^4, and there is node $k_{j'} \neq k_j$ sending flow to r on a path whose first edge has endpoint y_i^2 or y_i^6. Note that our assignment procedure is consistent and yields indeed a valid truth assignment if and only if there is no variable in conflict.

We now make a few claims on the structure of $\mathcal{S}$, that will be useful to show that no variable can be in conflict. Next fact follows from basic flow theory.

Fact 2. Without loss of generality, we can assume that the flow sent from terminals in C_1 (resp. C_2) to r, does not induce directed cycles.

Claim 4.5.2 *Without loss of generality, we can assume that every terminal sends flow to r on a path that contains exactly one node $y \in \bigcup_{i=1}^{p}\{y_i^1, y_i^3, y_i^5, y_i^7\}$.*

We defer the proof of this claim which is central to the remaining proof to the end. Let G_i be the subgraph of G induced by the nodes $\{r, y_i^1, \ldots, y_i^7\}$, and let χ_i be the total cost of the capacity that $\mathcal{S}$ installs on the subgraph G_i. Note that, by Fact 1, the cost of $\mathcal{S}$ is $m \cdot M + \sum_{i=1}^{m} \chi_i$. We will use Claim 1 to give a bound on the value χ_i. To this aim, let n_i be the number of nodes k_j whose path P_1^j contains edges of G_i. Note that $0 \leq n_i \leq 3$, and each k_j contributes to exactly one n_i, for some $i = 1, \ldots, p$.

Claim 4.5.3 *We have $\chi_i \geq 8 + 2n_i$, with the inequality being strict if the variable x_i is in conflict.*

Claim 4.5.3 finishes our proof, since it implies that the cost of $\mathcal{S}$ is at least

$$m \cdot M + \sum_{i=1}^{p} \chi_i \geq m \cdot M + \sum_{i=1}^{p}(8 + 2n_i) = m \cdot M + 8p + 2m,$$

with the inequality being tight if and only if there is no variable in conflict. ■

4.6 Latency SAND

As described in Section **??**, there is a $\Omega(\log n)$ gap between the tree and graph version of f-SAND. This naturally raises the question of approximating f-SAND when the solution must be restricted to different topologies. In this section, we consider the f-SAND when the output topology must be a path. Since this variant of f-SANDis not easy to solve on a tree, it is not clear how to solve it using tree metrics.

Definition 4.6.1 *In the* latency-f-SAND *problem, we are given an instance of* f-SAND*, but require the output to be a path with the root r as one of its endpoints. Our goal is output a minimum cost path, where the cost of an edge is $w_e \cdot$ (load on e). The load on an edge is the maximum number of nodes of one color it separates from the root.*

We assume that the lengths are integers and polynomially bounded in the input and give a time-indexed length formulation for this problem. This linear programming formulation was introduced by Chakrabarty and Swamy [31] for orienteering problems.

The Linear Programming Formulation for Latency-f-SAND

$$\begin{aligned}
\min \quad & \sum_{j,t} t \cdot x_{j,t} && && (\text{LP}^b_{\mathcal{P}})\\
\text{s.t.} \quad & \sum_{t} x_{j,t} \geq 1 && \forall j \in [m] && (4.2)\\
& \sum_{P \in \mathcal{P}_{b\cdot t}} z_{P,t} \leq 1 && \forall t \in [T] && (4.3)\\
& \sum_{P \in \mathcal{P}_{b\cdot t} : j \in P} z_{P,t} \geq \sum_{t' \leq t} x_{j,t'} && \forall j \in [m], t \in [T] && (4.4)\\
& x, z \geq 0
\end{aligned}$$

We assume without loss of generality, that $|C_i| = m$ for all $i \in [k]$. $\mathcal{P}_t$ denotes the set of paths of weight at most t starting from the root. Since the lengths are polynomially bounded, we can contain a variable for each possible length (we denote T to be the maximum possible length). We use $j \in P_t$ to indicate that the path P_t contains j terminals of each color. The variable $x_{j,t}$ indicates that we have seen j terminals of each color by time t and $z_{P,t}$ indicates that we use path P to visit the terminals at time t.

Lemma 4.6.2 *The linear program $LP^b_{\mathcal{P}}$ is a relaxation of Latency-*f-SAND*for $b \geq 1$.*

Proof: We show that the contraints and objective are valid for any feasible solution to Latency-f-SAND.

- Constraint 4.2 ensures that j terminals of each color are covered at some given time period, for every $j \in [m]$.
- Constraint 4.3 ensures that only one path is (fractionally) picked for each time period t.
- Constraint 4.4 indicates that we must have picked a path P that covers j terminals by time t if $\sum_{t' \leq t} x_{j,t'} = 1$.
- The objective function correctly captures the cost of the path. For an integer solution, $x_{j,t} = 1$ indicates that time t is the first time j terminals of each color are present in the path. Thus the objective counts the prefix length t^1 corresponding to where $x_{1,t^1} = 1$ in all m of the terms, the next prefix of length $t^2 - t^1$ in $m - 1$ of them and so on. This accurately accounts for the loads in these segments of the path according to the objective function in f-SAND. Finally, $b \geq 1$ only allows the paths to be of lengths longer by a factor of b so keeps the optimal solution feasible.

■

First, we can relax the above LP by replacing $\mathcal{P}_t$ with $\mathcal{T}_t$ which is the set of all trees of size at most t. This is a relaxation as $\mathcal{P}_t \subseteq \mathcal{T}_t$. Lemma 4.6.3, shows that we can round $\text{LP}^b_{\mathcal{T}}$ to get a $O(b)$ approximation to latency-f-SAND.

Lemma 4.6.3 *Given a fractional solution (x, z) to $LP^b_{\mathcal{T}}$, we can round it to a solution to latency-*f-SAND *with cost at most $O(b)$ times the cost of $LP^b_{\mathcal{T}}$.*

We defer the proof to the appendix due to space constraints but briefly sketch the argument. Roughly, we sample the trees at geometric intervals and "eulerify" them to produce a solution whose cost is not too much larger than the LP-objective.

Despite, being able to round the LP, we cannot hope to solve it effeciently due to the exponential number of variables in the primal. We will use the dual to obtain a solution to a relaxed version of

the primal.

$$\max \quad \sum_j \alpha_j - \sum_t \beta_t \qquad \text{(Dual}^b_{\mathcal{P}}\text{)}$$

$$\text{s.t.} \qquad \alpha_j \leq t + \sum_{t' \geq t} \theta_{j,t'} \qquad \forall j, t \tag{4.5}$$

$$\sum_{j \in P} \theta_{j,t} \leq \beta_t \qquad \forall t, P \in \mathcal{P}_{bt} \tag{4.6}$$

$$\alpha, \beta, \theta \geq 0. \tag{4.7}$$

Following [31] it is suffient that an "approximate separation oracle" in the sense of Lemma 4.6.4 is sufficient to compute an optimal solution to $\text{LP}^b_{\mathcal{T}}$.

Lemma 4.6.4 *Given a solution (α, β, θ), we can show that either (α, β, θ) is a solution to Dual$^1_{\mathcal{T}}$ or find a violated inequality for (α, β, θ) for Dual$^b_{\mathcal{T}}$ for $b = O(\log^2 k \log n)$.*

Once again, we defer the proof to the appendix, but sketch the argument. To efficiently separate, we observe that constraint 4.6 can be recast as a covering Steiner tree problem. Using approximation algorithms for this problem, we find a violated inequality for a (stronger) constraint. This results in the "approximate separation oracle".

Theorem 4.6.5 *There exists a $O(\log^2 k \log n)$ approximation to the Latency-SAND problem.*

Proof: Combining Lemma 3.2 of [31] with Lemma 4.6.4, we can now compute an ϵ-additive optimal solution to $\text{LP}^b_{\mathcal{T}}$ for $b = O(\log^2 k \log n)$. Using Lemma 4.6.3, we then achieve an $O(b)$ approximation for our problem. ■

4.7 Open Problems

The following is the main open question in this section:

Problem 4.7.1 *Does there exist an $O(1)$-approximation algorithm for the* f-SAND *problem.*

Although standard LP-based approaches seem to fail in providing a constant factor approximation, the worst known integrality gap example we are aware of yields a (trivial) lower bound of 2 on the integrality gap of (IP) for f-SAND. A related open question is if there is an instance of f-SAND for which the integrality gap of (IP) is greater than 2. Subsequent to the work mentioned above, "Mucha and Marcin" [124] in ongoing work showed an approximation ratio of $4/3$ for f-SAND when $k = 2$ and a 2-approx for $k = 3$.

Chapter 5

Online Bin Packing with Recourse

5.1 Introduction

An instance $\mathcal{I}$ of the BIN PACKING problem consists of a list of n items of sizes $s_1, s_2, \ldots, s_n \in [0, 1]$. The objective of a bin packing algorithm is to pack the items of $\mathcal{I}$ into a small number of unit-sized bins. Let $OPT(\mathcal{I})$ be the minimum number of unit-sized bins needed to pack all these items. This classic NP-hard optimization problem has been studied since the 1950s, with thousands of papers addressing this problem, variations and generalizations; see [86, Chapter 2] for an early survey, and [41] for a more up-to-date one. Much of this work (e.g. [17, 84, 98, 114, 127, 128, 131, 137, 140, 142]), starting with the seminal work of Ullman [136], studies the *online* setting, where items arrive sequentially and must be packed into bins immediately and irrevocably. However, the online problem is strictly harder than the offline version, due to the lack of information about the future: while the offline setting can be approximated to within a small additive term of $O(\log OPT)$ (see, e.g. [85]), in the online setting there is at least a 1.5403-multiplicative gap between the algorithm and OPT in the worst case, even as $OPT \to \infty$ [17].

Given the wide applicability of the online problem, researchers have considered the problem where a "small" number of repackings is allowed. I.e., how well can we perform if we allow items to be moved between bins when new items arrive and old ones depart? Clearly, some repacking is necessary; to make this question non-trivial, we demand bounded "recourse", i.e., items should be moved sparingly. Formally, a *fully-dynamic* BIN PACKING *algorithm* maintains at every time t, a feasible solution to the BIN PACKING instance $\mathcal{I}_t$ given by items inserted and not yet deleted until time t. Every item i has a size $s_i \in [0, 1]$, and a *movement cost* c_i which the algorithm pays every time item i is moved between bins.

Definition 5.1.1 *A fully-dynamic algorithm $\mathcal{A}$ has (i) an* asymptotic competitive ratio *(a.c.r) α, (ii)* additive term *a and (iii)* recourse *β, if at each time t it packs the instance $\mathcal{I}_t$ using at most $\alpha \cdot OPT(\mathcal{I}_t) + a$ bins, with a independent of $OPT(\mathcal{I}_t)$, while the total movement cost until time t is at most $\beta \cdot \sum_{i=1}^{t} c_i$. If at each time t algorithm $\mathcal{A}$ incurs at most $\beta \cdot c_t$ movement cost, where c_t is the cost of the element added or deleted at time t, we say algorithm $\mathcal{A}$ has* worst case *recourse β, otherwise we say it has* amortized *recourse β.*

The main goal of this chapter is to understand this question.

Problem 5.1.2 *What is the optimal trade-offs between having low asymptotic competitive ratios (a.c.r), additive terms, and recourse?*

5.2 Related Works

Gambosi et al. [65, 66] were the first to study dynamic BIN PACKING algorithms. They gave a $4/3$-a.c.r algorithm for the insertion-only setting which only moves every item a constant number of times throughout the algorithm's run, which in our terminology translates to constant amortized recourse for general movement costs. For unit movement costs, Ivković and Lloyd [91] and Balogh et al. [16, 18] gave lower bounds of $4/3$ and 1.3871 on the a.c.r of algorithms with constant recourse, respectively; Balogh et al. [18] also presented an algorithm with a.c.r tending to $3/2$ as the worst-case number of movements increases to infinity. In 2009, following Sanders et al. [129] who studied makespan minimization with bounded recourse, motivated by questions in sensitivity analysis, Epstein et al. [52] re-introduced the dynamic BIN PACKING problem for the case where the movement cost c_i of each item equals its size s_i (the *weight cost* setting).[1] They showed that bounded (though exponential in ϵ^{-1}) recourse suffices to maintain a solution with a.c.r $(1+\varepsilon)$. This was improved by Jansen and Klein [95] who showed that for insertion-only algorithms, a polynomial dependence on ε^{-1} suffices; Berndt et al. [25] showed the same for fully-dynamic algorithms. They further showed that any $(1+\varepsilon)$ a.c.r algorithm must have worst-case recourse of $\Omega(\varepsilon^{-1})$. While these give strong, nearly-tight results in the case of weight costs $c_j = s_j$, the unit costs $c_i = 1$ and general costs cases are not so well understood.

5.3 Our Results

We fully characterize the recourse to asymptotic competitive ratio trade-off for fully-dynamic BIN PACKING under general movement costs, unit movement costs and weight costs. Our results are summarized in the following five theorems. In all these results, we use the term *online* BIN PACKING to refer to the classical insertion-only model without any repacking, and *fully-dynamic* BIN PACKING *(with recourse)* to refer to the model with both insertions and deletions where we can repack items.

General Costs

The results of [25, 52, 95] show that in the weight cost model a little recourse can circumvent the negative effects of online arrivals, and allow for arbitrarily good a.c.r. Does such a claim hold even for general costs? Our first result shows that even allowing for repacking, the fully-dynamic BIN PACKING problem is no easier than the arrival-only online problem (with no repacking).

[1] Some work in this area uses the term *migration factor* instead of *recourse* in the setting of weight movement cost; we use the term recourse for brevity, and to emphasize the broader set of movement costs considered here.

Theorem 5.3.1 (Fully Dynamic as Hard as Online) *Any fully-dynamic* BIN PACKING *algorithm with bounded recourse under general movement costs has asymptotic competitive ratio at least as high as that of any online* BIN PACKING *algorithm. Given current bounds, this is at least* 1.54037.

Now the concern is: since elements can both arrive and depart, is it conceivable that the fully-dynamic model is *harder* than the online one, even allowing for recourse? We show this is likely not the case, as we can almost match the a.c.r of the current-best algorithm for online BIN PACKING.

Theorem 5.3.2 (Fully Dynamic as Easy as Online) *Any algorithm in the Super Harmonic family of algorithms can be implemented in the fully-dynamic setting with constant recourse under general movement costs, yielding the same a.c.r. This implies an algorithm with a.c.r of* 1.58889 *using [131].*

The current best online BIN PACKING algorithm, [84], is not from the Super Harmonic family, but is closely related to them. It has an a.c.r of 1.5815, so our results for fully-dynamic BIN PACKING are within a hair's width of the best bounds known for online BIN PACKING.

Unit Costs

We now consider the most natural of all movement costs, that of *unit costs*; i.e., $c_i = 1$ for all items i. For this model we give tight upper and lower bounds. Let $\alpha = 1 - \frac{1}{W_{-1}(-2/e^3)+1} \approx 1.3871$ (here W_{-1} is the lower real branch of the Lambert W-function [43]). Balogh et al. [16] showed α is a lower bound on the a.c.r of all fully-dynamic BIN PACKING algorithms with constant recourse. We present a simpler proof of this lower bound, and show that surpassing this a.c.r requires either *polynomial* additive term or recourse (or both). Moreover, we present an algorithm proving α is tight for this problem.

Theorem 5.3.3 (Unit Costs Tight Bounds) *For any $\varepsilon > 0$, there exists a fully-dynamic* BIN PACKING *algorithm with a.c.r $(\alpha + \varepsilon)$, additive term $O(\varepsilon^{-2})$ and amortized recourse $O(\varepsilon^{-2})$ under unit movement costs. Conversely, for any fully-dynamic* BIN PACKING *algorithm with a.c.r $(\alpha - \varepsilon)$, the product of the additive term and the amortized recourse cost must be $\Omega(\varepsilon^4 \cdot n)$ under unit movement costs.*

These complementary results give us a sharp threshold on the a.c.r of any algorithm for the unit costs model. This is the technical heart of the paper: we show both upper and lower bounds using linear programming techniques. We give a linear program that completely captures the performance of the algorithm. Indeed, we first use this LP as a *gap-revealing* LP, to prove that a certain family of instances give an a.c.r of at least $\approx \alpha$. Then we use it as a *factor-revealing* LP to show that our algorithm achieves an a.c.r at most $\approx \alpha$.

Weight Costs

Finally, we give an extension of the already strong results known for the weight cost model. We show that the lower bound known in the worst-case recourse model extends to the amortized model as well, for which it is tight.

Theorem 5.3.4 (Weight Costs Tight Bounds) *For any $\varepsilon > 0$, there exists a $(1 + \varepsilon)$-a.c.r algorithm with constant additive term and $O(\varepsilon^{-1})$ amortized recourse under weight movement costs.*

Conversely, there exist infinitely many $\varepsilon > 0$ such that any $(1+\varepsilon)$-a.c.r algorithm requires $\Omega(\varepsilon^{-1})$ amortized recourse under weight movement costs.

Note that the hardness result of the last theorem was only known for *worst-case* recourse ([25]); this previous lower bound consists of a hard instance which effectively disallowed any recourse, while lower bounds against amortized recourse can of course not do so. In the context of sensitivity analysis, our lower bound shows that in order to maintain a near-optimal, i.e. a $(1+\varepsilon)$-asymptotically competitive solution at least $\Omega(\varepsilon^{-1})$ volume times the volume of items added and removed must be moved, even in an amortized sense.

5.4 Unit Movement Costs

We consider the natural unit movement costs model. First, we show that an a.c.r better than $\alpha = 1 - \frac{1}{W_{-1}(-2/e^3)+1} \approx 1.3871$ implies either polynomial additive term or recourse. Next, we give tight $(\alpha+\epsilon)$-a.c.r algorithms, with both additive term and recourse polynomial in ϵ^{-1}. Our key idea is to use an LP that acts both as a *gap-revealing LP* for a lower bound, and also as a *factor-revealing LP* (as well as an inspiration) for our algorithm.

5.4.1 Impossibility results

Alternating between two instances, where both have $2n-4$ items of size $1/n$ and one instance also has 4 items of size $1/2 + 1/2n$, we get that any $\left(\frac{3}{2}-\varepsilon\right)$-competitive online bin packing algorithm must have $\Omega(n)$ recourse. However, this only rules out algorithms with a *zero* additive term. Balogh et al. [16] showed that any $O(1)$-recourse algorithm (under unit movement costs) with $o(n)$ additive term must have a.c.r at least $\alpha \approx 1.3871$. We strengthen both this impossibility result by showing the need for a large additive term or recourse to achieve any a.c.r below α. Specifically, arbitrarily small polynomial additive terms imply near-linear recourse. Our proof is shorter and simpler than in [16]. As an added benefit, the LP we use to bound the competitive ratio will inspire our algorithm in the next section.

Theorem 5.4.1 (Unit Costs: Lower Bound) *For any $\varepsilon > 0$ and $\frac{1}{2} > \delta > 0$, for any algorithm $\mathcal{A}$ with a.c.r $(\alpha - \varepsilon)$ with additive term $o(\varepsilon \cdot n^{\delta})$, there exists an dynamic bin packing input with n items on which $\mathcal{A}$ uses recourse at least $\Omega(\varepsilon^2 \cdot n^{1-\delta})$ under unit movement cost.*

The Instances: The set of instances is the a natural one, and was also considered by [16]. Let $1/2 > \delta > 0$, $c = 1/\delta - 1$, and let $B \geq 1/\varepsilon$ be a large integer. Our hard instances consist of *small* items of size $1/B^c$, and *large* items of size ℓ for all sizes $\ell \in \mathcal{S}_\varepsilon \triangleq \{\ell = 1/2 + i \cdot \varepsilon \mid i \in \mathbb{N}_{>0},\ \ell \leq 1/\alpha\}$. Specifically, input $\mathcal{I}_s$ consists of $\lfloor B^{c+1} \rfloor$ small items, and for each $\ell \in \mathcal{S}_\varepsilon$, the input $\mathcal{I}_\ell$ consists of $\lfloor B^{c+1} \rfloor$ small items followed by $\lfloor \frac{B}{1-\ell} \rfloor$ size-ℓ items. The optimal bin packings for $\mathcal{I}_s$ and $\mathcal{I}_\ell$ require precisely $OPT(\mathcal{I}_s) = B$ and $OPT(\mathcal{I}_\ell) = \lceil \frac{B}{1-\ell} \rceil$ bins respectively. Consider any fully-dynamic bin packing algorithm $\mathcal{A}$ with limited recourse and bounded additive term. When faced with input $\mathcal{I}_s$, algorithm $\mathcal{A}$ needs to distribute the small items in the available bins almost uniformly. And if this input is extended to $\mathcal{I}_\ell$ for some $\ell \in \mathcal{S}_\varepsilon$, algorithm $\mathcal{A}$ needs to move many small items to

accommodate these large items (or else use many new bins). Since $\mathcal{A}$ does not know the value of t beforehand, it cannot "prepare" simultaneously for all large sizes $\ell \in \mathcal{S}_\varepsilon$.

As a warm-up we show that the linear program (LP$_\varepsilon$) below gives a lower bound on the *absolute* competitive ratio α_ε of any algorithm $\mathcal{A}$ with no recourse. Indeed, instantiate the variables as follows. On input $\mathcal{I}_s$, let N_x be the number of bins in which $\mathcal{A}$ keeps free space in the range $[x, x+\varepsilon)$ for each $x \in \{0\} \cup \mathcal{S}_\varepsilon$. Hence the total volume packed is at most $N_0 + \sum_{x \in \mathcal{S}_\varepsilon} (1-x) \cdot N_x$. This must be at least $Vol(\mathcal{I}_s) \geq B - 1/B^c$, implying constraint (Vol$_\varepsilon$). Moreover, as $OPT(\mathcal{I}_s) = B$, the α_ε-competitiveness of $\mathcal{A}$ implies constraint (small$_\varepsilon$). Now if instance $\mathcal{I}_s$ is extended to $\mathcal{I}_\ell$, since $\mathcal{A}$ moves no items, these ℓ-sized items are placed either in bins counted by N_x for $x \geq \ell$ or in new bins. Since $\mathcal{A}$ is α_ε-competitive and $OPT(\mathcal{I}_\ell) \leq \lceil \frac{B}{1-\ell} \rceil$ we get constraint (CR$_\varepsilon$). Hence the optimal value of (LP$_\varepsilon$) is a valid lower bound on the competitive ratio α_ε.

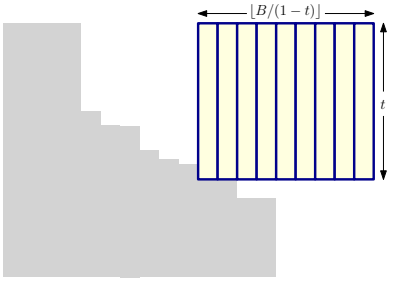

Figure 5.1: A packing of instance $\mathcal{I}_\ell$. The ℓ-sized items (in yellow) are packed "on top" of the small items (in grey).

$$
\begin{aligned}
&\text{minimize } \alpha_\varepsilon && && && (\text{LP}_\varepsilon)\\
&\text{s.t. } N_0 + \textstyle\sum_{x \in \mathcal{S}_\varepsilon} (1-x) \cdot N_x && \geq B - 1/B^c && && (\text{Vol}_\varepsilon)\\
&\quad N_0 + \textstyle\sum_{x \in \mathcal{S}_\varepsilon} N_x && \leq \alpha_\varepsilon \cdot B && && (\text{small}_\varepsilon)\\
&\quad N_0 + \textstyle\sum_{x \in \mathcal{S}_\varepsilon, x \leq \ell - \varepsilon} N_x + \left\lfloor \frac{B}{1-\ell} \right\rfloor && \leq \alpha_\varepsilon \cdot \left\lceil \frac{B}{1-\ell} \right\rceil && \forall \ell \in \mathcal{S}_\varepsilon && (\text{CR}_\varepsilon)\\
&\quad N_x \geq 0
\end{aligned}
$$

The claimed lower bound on the a.c.r of recourse-less algorithms follows from Lemma 5.4.2.

Lemma 5.4.2 *The optimal value $\alpha^\star_\varepsilon$ of* (LP$_\varepsilon$) *satisfies $\alpha^\star_\varepsilon \in [\alpha - O(\varepsilon), \alpha + O(\varepsilon)]$.*

To extend the above argument to the fully-dynamic case, we observe that any solution to (LP$_\varepsilon$) defined by packing of $\mathcal{I}_s$ as above must satisfy some constraint (CR$_\varepsilon$) for some $\ell \in \mathcal{S}_\varepsilon$ with equality, implying a competitive ratio of at least α_ϵ. Now, to beat this bound, a fully-dynamic algorithm must move at least ϵ volume of small items from bins which originally had less than $x - \epsilon$ free space. As small items have size $1/B^c = 1/n^\delta$, this implies that $\Omega(\epsilon n^\delta)$ small items must be moved for every bin affected by such movement. This argument yields Lemma 5.4.3, which together with Lemma 5.4.2 implies Theorem 5.4.1.

Lemma 5.4.3 *For all $\varepsilon > 0$ and $\frac{1}{2} > \delta > 0$, if $\alpha^\star_\varepsilon$ is the optimal value of* (LP$_\varepsilon$)*, then any fully-dynamic bin packing algorithm $\mathcal{A}$ with a.c.r $\alpha^\star_\varepsilon - \varepsilon$ and additive term $o(\varepsilon^2 \cdot n^\delta)$ has recourse $\Omega(\varepsilon^2 \cdot n^{1-\delta})$ under unit movement costs.*

5.4.2 Matching Algorithmic Results

As mentioned earlier, LP$_\varepsilon$ also guides our algorithm. For the rest of this section, items smaller than ε are called *small*, and the rest are *large*. Items of size $s_i > 1/2$ are *huge*.

To begin, imagine a total of B volume of small items come first, followed by large items. Inspired by the LP analysis above, we pack the small items such that an N_ℓ/B fraction of bins have ℓ free space for all $\ell \in \{0\} \cup \mathcal{S}_\varepsilon$, where the N_ℓ values are near-optimal for (LP$_\varepsilon$). We call the space profile used by the small items the "curve"; see Figure 5.1. In the LP analysis above, we showed that this packing can be extended to a packing with a.c.r $\alpha+O(\varepsilon)$ if $B/(1-\ell)$ items of size ℓ are added. But what if large items of *different* sizes are inserted and deleted? In §5.4.2we show that this approach retains its $(\alpha + O(\epsilon))$-a.c.r in this case too, and outline a linear-time algorithm to obtain such a packing.

The next challenge is that the small items may also be inserted and deleted. In §5.4.2 we show how to dynamically pack the small items with bounded recourse, so that the number of bins with any amount of free space $f \in \mathcal{S}_\varepsilon$ induce a near-optimal solution to LP$_\varepsilon$.

Finally, in §5.4.2 we combine the two ideas together to obtain our fully-dynamic algorithm.

LP$_\varepsilon$ as a Factor-Revealing LP

In this section we show how we use the linear program LP$_\varepsilon$ to analyze and guide the following algorithm: pack the small items according to a near-optimal solution to LP$_\varepsilon$, and pack the large items near-optimally "on top" of the small items.

To analyze this approach, we first show it yields a good packing if all large items are huge and have some common size $\ell > 1/2$. Consider an instance $\mathcal{I}_s$ consisting solely of small items with total volume B, packed using N_x bins with gaps in the range $[x, x+\varepsilon)$ for all $x \in \{0\} \cup \mathcal{S}_\varepsilon$, where $\{\alpha_\varepsilon, N_0, N_x : x \in \mathcal{S}_\varepsilon\}$ form a feasible solution for (LP$_\varepsilon$). We say such a packing is α_ε-*feasible*. By the LP's definition, any α_ε-feasible packing of small items can be extended to an α_ε-competitive packing for any extension $\mathcal{I}_\ell$ of $\mathcal{I}_s$ with $\ell \in \mathcal{S}_\varepsilon$, by packing the size ℓ items in the bins counted by N_x for $x \geq \ell$ before using new bins. In fact, this solution satisfies a similar property for any extension $\mathcal{I}_\ell^k$ obtained from $\mathcal{I}_s$ by adding *any* number k of items all of size ℓ. (Note that $\mathcal{I}_\ell$ is the special case of $\mathcal{I}_\ell^k$ with $k = \lfloor B/(1-\ell) \rfloor$.)

Lemma 5.4.4 (Huge Items of Same Size) *Any α_ε-feasible packing of small items of $\mathcal{I}_s$ induces an α_ε-competitive packing for all extensions $\mathcal{I}_\ell^k$ of $\mathcal{I}_s$ with $\ell > 1/2$ and $k \in \mathbb{N}$.*

Now, to pack the large items of the instance $\mathcal{I}$, we first create a similar new instance $\mathcal{I}'$ whose large items are all huge items. To do so, we first need the following observation.

Observation 5.4.5 *For any input $\mathcal{I}$ made up of solely large items and function $f(\cdot)$, a packing of $\mathcal{I}$ using at most $(1+\varepsilon) \cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ bins has all but at most $2\varepsilon \cdot OPT(\mathcal{I}) + 2f(\varepsilon^{-1}) + 3$ of its bins containing either no large items or being more than half filled by large items.*

Consider a packing $\mathcal{P}$ of the large items of $\mathcal{I}$ using at most $(1+\varepsilon) \cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ bins. By Observation 5.4.5, at most $2\varepsilon \cdot OPT(\mathcal{I}) + O(f(\varepsilon^{-1}))$ bins in $\mathcal{P}$ are at most half full. We use these bins when packing $\mathcal{I}$. For each of the remaining bins of $\mathcal{P}$, we "glue" all large items occupying the same bin into a single item, yielding a new instance $\mathcal{I}'$ with only huge items, with any packing of $\mathcal{I}'$ "on top" of the small items of $\mathcal{I}$

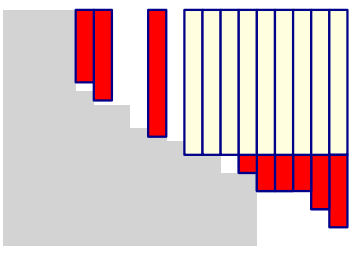

Figure 5.2: A packing of instance $\mathcal{I}'$. Large items are packed "on top" of the small items (in grey). Parts of large items not in the instance $\mathcal{I}_\ell^k$ are indicated in red.

trivially inducing a similar packing of $\mathcal{I}$ with the same number of bins. We pack the huge items of $\mathcal{I}'$ on top of the curve greedily, repeatedly packing the smallest such item in the fullest bin which can accommodate it. Now, if we imagine we remove and decrease the size of some of these huge items, this results in a new (easier) instance of the form $\mathcal{I}_\ell^k$ for some k and $\ell > 1/2$, packed in no more bins than $\mathcal{I}'$ (see Figure 5.2). By Lemma 5.4.4, the number of bins used by our packing of $\mathcal{I}'$ (and of $\mathcal{I}$) is at most

$$\alpha_\varepsilon \cdot OPT(\mathcal{I}_\ell^k) \leq \alpha_\varepsilon \cdot OPT(\mathcal{I}') \leq (\alpha_\varepsilon + O(\epsilon)) \cdot OPT(\mathcal{I}) + O(f(\varepsilon^{-1})).$$

To obtain worst-case bounds, we extend this idea as follows: we near-optimally pack large items of size exceeding $1/4$. Given this packing of small and large items, we pack items of size in the range $(\varepsilon, 1/4]$ (which we refer to as *medium* items) using first-fit so that we only open a new bin if all bins are at least $3/4$ full. If we open a new bin, by a volume bound this packing has a.c.r $4/3 < \alpha + O(\epsilon)$. If we don't open a new bin, this packing is $\alpha + O(\epsilon)$ competitive against an easier instance (obtained by removing the medium items), and so we are $\alpha + O(\epsilon)$ competitive. These ideas underlie the following two theorems, a more complete proof of which appears in §C.1.2.

Theorem 5.4.6 *An α_ε-feasible packing of the small items of an instance $\mathcal{I}$ can be extended into a packing of all of $\mathcal{I}$ using at most $(\alpha_\epsilon + O(\varepsilon)) \cdot OPT(\mathcal{I}) + O(\varepsilon^{-2})$ bins in linear time for any fixed ϵ.*

Theorem 5.4.7 *For a dynamic instance $\mathcal{I}_t$, given a fully-dynamic $(1+\epsilon)$-a.c.r algorithm with additive term $f(\epsilon^{-1})$ for items of size greater than $1/4$ in $\mathcal{I}_t$, and a fully-dynamic α_ε-feasible packing of its small items, one can maintain a packing with a.c.r $(\alpha_\epsilon + O(\varepsilon))$ with additive term $O(f(\epsilon^{-1}))$. This can be done using worst-case recourse*

1. *$O(\varepsilon^{-1})$ per item move in the near-optimal fully-dynamic packing of the items of size $> 1/4$,*
2. *$O(\epsilon^{-1})$ per insertion or deletion of medium items, and*
3. *$O(1)$ per item move in the α_ϵ-feasible packing of the small items.*

It remains to address the issue of maintaining an α_ε-feasible packing of small items dynamically using limited recourse.

Dealing With Small Items: "Fitting a Curve"

We now consider the problem of packing ε-small items according to an approximately-optimal solution of (LP_ε). We abstract the problem thus.

[Bin curve-fitting] Given a list of bin sizes $0 \leq b_0 \leq b_1 \leq \ldots, b_K \leq 1$ and relative frequencies $f_0, f_1, f_2, \ldots, f_K$, such that $f_x \geq 0$ and $\sum_{x=0}^{K} f_x = 1$, an algorithm for the *bin curve-fitting problem* must pack a set of m of items with sizes $s_1, \ldots, s_m \leq 1$ into a minimal number of bins N such that for every $x \in [0, K]$ the number of bins of size b_x that are used by this packing lie in $\{\lfloor N \cdot f_x \rfloor, \lceil N \cdot f_x \rceil\}$.

If we have $K = 0$ with $b_0 = 1$ and $f_0 = 1$, we get standard BIN PACKING. We want to solve the problem only for (most of the) small items, in the fully-dynamic setting. We consider

the special case with relative frequencies f_x being multiples of $1/T$, for $T \in \mathbb{Z}$; e.g., $T = O(\varepsilon^{-1})$. Our approach is inspired by the algorithm of [95], and maintains bins in increasing sorted order of item sizes. The number of bins is always an integer product of T. Consecutive bins are aggregated into *clumps* of exactly T bins each, and clumps aggregated into $\Theta(\varepsilon^{-1})$ *buckets* each. Formally, each clump has T bins, with $f_x \cdot T \in \mathbb{N}$ bins of size b_x for $x = 0, \ldots, K$. The bins in a clump are ordered according to their capacity b_x, so each clump looks like its target curve. Each bucket except the last consists of some $s \in [1/\varepsilon, 3/\varepsilon]$ consecutive clumps (the last bucket may have fewer than $1/\epsilon$ clumps). See Figure 5.3. For each bucket, all bins except those in the last clump are full to within an additive ε.

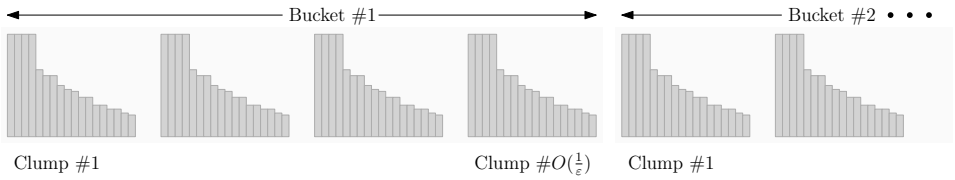

Figure 5.3: Buckets have $O(\varepsilon^{-1})$ clumps, clumps have T bins.

Inserting an item adds it to the correct bin according to its size. If the bin size becomes larger than the target size for the bin, the largest item overflows into the next bin, and so on. Clearly this maintains the invariant that we are within an additive ε of the target size. We perform $O(T/\epsilon)$ moves in the same bucket; if we overflow from the last bin in the last clump of the bucket, we add a new clump of T new bins to this bucket, etc. If a bucket contains too many clumps, it splits into two buckets, at no movement cost. An analogous (reverse) process happens for deletes. Loosely, the process maintains that on average the bins are full to within $O(\varepsilon)$ of the target fullness – one loss of ε because each bin may have ε space, and another because an $O(\varepsilon)$ fraction of bins have no guarantees whatsoever.

We now relate this process to the value of LP_ε. We first show that setting $T = O(\varepsilon^{-1})$ and restricting to frequencies which are multiples of ε does not hurt us. Indeed, for us, $b_0 = 1$, and $b_x = (1 - x)$ for $x \in \mathcal{S}_\varepsilon$. Since ($\text{LP}_\varepsilon$) depends on the total volume B of small items, and f_x may change if B changes, it is convenient to work with the normalized LP (LPnew_ε). Now n_x can be interpreted as just being proportional to number of bins of size b_x, and we can define $f_x = n_x / \sum_x n_x$. However, we also need each f_x to be an integer multiple of $1/T$ for some integer $T = O(\varepsilon^{-1})$. We achieve this by slightly modifying the LP solution, obtaining the following.

Lemma 5.4.8 (Multiples of ε) *For any optimal solution $\{n_x\}$ to (LPnew_ε) with objective value α_ε, we can construct in linear time a solution $\{\tilde{n}_x\} \subseteq \varepsilon \cdot \mathbb{N}$ with objective value $\alpha_\varepsilon + O(\varepsilon)$.*

Using a near-optimal solution to (LP_ε) as in the above lemma, we obtain a bin curve-fitting instance with $T = O(\varepsilon^{-1})$. Noting that if we ignore the last bucket's $O(T/\varepsilon^{-1}) = O(\varepsilon^{-2})$ bins in our solution, we obtain an $(\alpha^\star_\varepsilon + O(\varepsilon))$-feasible packing (with additive term $O(\varepsilon^{-2})$) of the remaining items by our algorithm above, using $O(\varepsilon^{-2})$ worst-case recourse. We obtain the following.

Lemma 5.4.9 (Small Items Follow the LP) *Let $\varepsilon \leq 1/6$. Using $O(\varepsilon^{-2})$ worst-case recourse we can maintain packing of small items such that the content of all but $O(\varepsilon^{-2})$ designated bins in this packing form an $(\alpha^\star_\varepsilon + O(\varepsilon))$-feasible packing.*

Our Algorithm

From Lemma 5.4.8 and Lemma 5.4.9, we can maintain an $(\alpha^\star_\varepsilon + O(\varepsilon))$-feasible packing of the small items (that is, a packing of inducing a solution to (LP$_\varepsilon$) with objective value $(\alpha^\star_\varepsilon + O(\varepsilon))$), all while using $O(\varepsilon^{-2})$ worst-case recourse. From Theorem 5.4.6 and Theorem 5.4.7, using a $(1+\varepsilon)$-a.c.r packing of the large items one can extend such a packing of the small items into an $(\alpha^\star_\varepsilon + O(\varepsilon))$-approximate packing for $\mathcal{I}_t$, where $\alpha^\star_\varepsilon \leq \alpha + O(\varepsilon)$, by Lemma 5.4.2. It remains to address the recourse incurred by extending this packing to also pack the large items.

Amortized Recourse

Here we periodically recompute in linear time the extension guaranteed by Theorem 5.4.6. Dividing the time into epochs and lazily addressing updates to large items (doing nothing on deletion and opening new bins on insertion) for $\varepsilon \cdot N$ steps, where N is the number of bins we use at the epoch's start, guarantees a $(\alpha + O(\varepsilon))$-a.c.r throughout the epoch. As for this approach's recourse, we note that the number of large items at the end of an epoch of length $\epsilon \cdot N$ is at most $O(N/\epsilon)$, and so repacking them incurs $O(N/\varepsilon)/(\epsilon \cdot N) = O(\epsilon^{-2})$ amortized recourse.

Worst-Case Recourse

To obtain worst-case recourse we rely on the fully-dynamic $(1+\varepsilon)$-a.c.r algorithm of Berndt al. [25, Theorem 9] to maintain a $(1+\varepsilon)$-approximate packing of the sub-instance made of items of size exceeding $1/4$ using $\tilde{O}(\varepsilon^{-3})$ size cost recourse, and so at most $\tilde{O}(\varepsilon^{-3})$ item moves (as these items all have size $\Theta(1)$). By Theorem 5.4.7, our $(\alpha + O(\varepsilon))$-feasible solution for the small items of $\mathcal{I}_t$ can be extended dynamically to an $(\alpha + (O(\varepsilon)))$-a.c.r packing of all of $\mathcal{I}_t$, using $\tilde{O}(\varepsilon^{-4})$ worst-case recourse.

From the above discussions we obtain our main result – tight algorithms for unit costs.

Theorem 5.4.10 (Unit Costs: Upper Bound) *There is a polytime fully-dynamic* BIN PACKING *algorithm which achieves $\alpha + O(\varepsilon)$ a.c.r, additive term $O(\varepsilon^{-2})$ and $O(\varepsilon^{-2})$ amortized recourse, or additive term $poly(\varepsilon^{-1})$ and $\tilde{O}(\varepsilon^{-4})$ worst case recourse, under unit movement costs.*

5.5 General Movement Costs

We now consider the case of general movement costs, and show a close connection with the (arrival-only) online problem. We first show that the fully-dynamic problem under general movement costs cannot achieve a better a.c.r than the online problem. Next, we match the a.c.r of any SUPER-HARMONIC algorithm for the online problem in the fully-dynamic setting.

5.5.1 Matching the Lower Bounds for Online Algorithms

Formally, an *adversary process* $\mathcal{B}$ for the online BIN PACKING problem is an adaptive process that, depending on the state of the system (i.e., the current set of configurations used to pack the current set of items) either adds a new item to the system, or stops the request sequence. We say that

an adversary process shows a lower bound of c for online algorithms for BIN PACKING if for any online algorithm $\mathcal{A}$, this process starting from the empty system always eventually reaches a state where the a.c.r is at least c.

Let $\beta \geq 2$. Any adversary process $\mathcal{B}$ showing a lower bound of c for the a.c.r of any online BIN PACKING algorithm can be converted into a fully-dynamic BIN PACKING instance with general movement costs such that any fully-dynamic BIN PACKING algorithm with amortized recourse at most β must have a.c.r at least c.

Such a claim is simple for *worst-case* recourse. Indeed, given a recourse bound β, set the movement cost of the i-th item to be $(\beta + \varepsilon)^{n-i}$ for $\varepsilon > 0$. When the i-th item arrives we cannot repack *any* previous item because their movement costs are larger by a factor of $> \beta$. So this is a regular online algorithm. The argument fails, however, if we allow amortization.

To construct our lower bound instance, we start with an adversary process and create a dynamic instance as follows. Each subsequent arriving item has exponentially decreasing (by a factor β) movement cost. When the next item arrives, our algorithm could move certain existing items. These items would have much higher movement cost than the arriving item, and so, this process cannot happen too often. Whenever this happens, we *reset* the state of the process to an earlier time and remove all jobs arriving in the intervening period. This will ensure that the algorithm always behaves like an online algorithm which has not moved any of the existing items. Since it cannot move jobs too often, the set of existing items which have not been moved by the algorithm grow over time. This idea allows us to show:

Corollary 5.5.1 *No fully-dynamic* BIN PACKING *algorithm with bounded recourse under general movement costs and $o(n)$ additive term is better than 1.54037-asymptotically competitive.*

5.5.2 (Nearly) Matching the Upper Bounds for Online Algorithms

We outline some of the key ingredients in obtaining an algorithm with competitive ratio nearly matching our upper bound of the previous section. The first is an algorithm to pack similarly-sized items.

[Near-Uniform Sizes] There exists a fully-dynamic BIN PACKING algorithm with constant worst case recourse which given items of sizes $s_i \in [1/k, 1/(k-1))$ for some integer $k \geq 1$, packs them into bins of which all but one contain $k-1$ items and are hence at least $1 - 1/k$ full. (If all items have size $1/k$, the algorithm packs k items in all bins but one.)

Lemma 5.5.2 readily yields a 2-a.c.r algorithm with constant recourse (see §C.2.2). We now discuss how to obtain 1.69-a.c.r based on this lemma and the HARMONIC algorithm [114].

The Harmonic Algorithm

The idea of packing items of nearly equal size together is commonly used in online BIN PACKING algorithms. For example, the HARMONIC algorithm [114] packs large items (of size $\geq \epsilon$) as in Lemma 5.5.2, while packing small items (of size $\leq \varepsilon$) into dedicated bins which are at least $1 - \varepsilon$ full on average, using e.g., FIRSTFIT. This algorithm uses $(1.69 + O(\varepsilon)) \cdot OPT + O(\varepsilon^{-1})$ bins [114]. Unfortunately, due to item removals, FIRSTFIT won't suffice to pack small items into nearly-full bins.

To pack small items in a dynamic setting we extend our ideas for the unit cost case, partitioning the bins into *buckets* of $\Theta(1/\epsilon)$ many bins, such that all but one bin in a bucket are $1 - O(\epsilon)$ full, and hence the bins are $1 - O(\epsilon)$ full on average. Since the size and cost are not commensurate, we maintain the small items in sorted order according to their *Smith ratio* (c_i/s_i). However, insertion of a small item can create a large cascade of movements throughout the bucket. We only move items to/from a bin once it has $\Omega(\epsilon)$'s worth of volume removed/inserted (keeping track of erased, or "ghost" items). A potential function argument allows us to show amortized $O(\epsilon^{-2})$ recourse cost for this approach, implying the following lemma, and by [114], a $(1.69 + O(\varepsilon))$-a.c.r algorithm with $O(\epsilon^{-2})$ recourse.

For all $\varepsilon \leq \frac{1}{6}$ there exists an asymptotically $(1 + O(\varepsilon))$-competitive bin packing algorithm with $O(\varepsilon^{-2})$ amortized recourse if all items have size at most ε.

Seiden's Super-Harmonic Algorithms

We now discuss our remaining ideas to match the bounds of any Super-Harmonic algorithm [131] in the fully-dynamic setting. A *Super-harmonic* (SH) algorithm partitions the unit interval $[0, 1]$ into $K + 1$ intervals $[0, \varepsilon], (t_0 = \varepsilon, t_1](t_1, t_2], \ldots, (t_{K-1}, t_K = 1]$. Small items (of size $\leq \varepsilon$) are packed into dedicated bins which are $1 - \varepsilon$ full. A large item has type i if its size is in the range $(t_{i-1}, t_i]$. The algorithm also colors items blue or red. Each bin contains items of at most two distinct item types i and j. If a bin contains only one item type, all its items are colored the same. If a bin contains two item types $i \neq j$, all type i items are colored blue and type j ones are colored red (or vice versa). The SH algorithm is defined by four sequences $(\alpha_i)_{i=1}^K, (\beta_i)_{i=1}^K, (\gamma_i)_{i=1}^K$, and a bipartite *compatibility graph* $\mathcal{G} = (V, E)$. A bin with blue (resp., red) type i items contains at most β_i (resp., γ_i) items of type i, and is *open* if it contains less than this many type i items. The compatibility graph $\mathcal{G} = (V, E)$ has vertex set $V = \{b_i \mid i \in [K]\} \cup \{r_j \mid j \in [K]\}$, with an edge $(b_i, r_j) \in E$ indicating blue items of type i and red items of type j are *compatible* and may share a bin. In addition, an SH algorithm must satisfy the following invariants.

(P1) The number of open bins is $O(1)$.

(P2) If n_i is the number of type-i items, the number of red type-i items is $\lfloor \alpha_i \cdot n_i \rfloor$.

(P3) If $(b_i, r_j) \in E$ (blue type i items and red type j items are compatible), there is no pair of bins with one containing only blue type i items and one containing only red type j items.

Appropriate choice of $(t_i)_{i=1}^{K+1}, (\alpha_i)_{i=1}^K, (\beta_i)_{i=1}^K, (\gamma_i)_{i=1}^K$ and $\mathcal{G}$ allows one to bound the a.c.r of any SH algorithm. (E.g., Seiden gives an SH algorithm with a.c.r 1.58889 [131].)

Simulating SH algorithms.

In a sense, SH algorithms ignore the exact size of large items, so we can take all items of some type and color. This extends Lemma 5.5.2 to pack at most β_i or γ_i of them per bin to satisfy Properties 1 and 2. The challenge is in maintaining Property 3: consider a bin with β_i blue type i items and γ_j type-j items, and suppose the type i items are all removed. Suppose there exists an open bin with items of type $i' \neq i$ compatible with j. If the movement costs of both type j and type i' items are significantly higher than the cost of the type i items, we cannot afford to place these groups

together, violating Property 3. To avoid such a problem, we use ideas from *stable matchings*. We think of groups of β_i blue type-i items and γ_j red type-j items as nodes in a bipartite graph, with an edge between these nodes if $\mathcal{G}$ contains the edge (b_i, r_j). We maintain a stable matching under updates, with priorities being the value of the costliest item in a group of same-type items packed in the same bin. The stability of this matching implies Property 3; we maintain this stable matching using (a variant of) the Gale-Shapley algorithm. Finally, relying on our solution for packing small items as in §5.5.2, we can pack the small items in bins which are $1 - \varepsilon$ full on average. Combined with Lemma C.2.2, we obtain following:

There exists a fully-dynamic BIN PACKING algorithm with a.c.r 1.58889 and constant additive term using constant recourse under general movement costs.

5.6 Size Movement Costs (Migration Factor)

In this section, we settle the optimal recourse to a.c.r tradeoff for size movement cost (referred to as *migration factor* in the literature); that is, $c_i = s_i$ for each item i. For worst case recourse in this model (studied in [25, 52, 95]), $poly(\epsilon^{-1})$ upper and lower bounds are known for $(1+\epsilon)$-a.c.r. algorithms [25], though the optimal tradeoff remains elusive. We show that for amortized recourse the optimal tradeoff is $\Theta(\epsilon^{-1})$ recourse for $(1+\epsilon)$-a.c.r.

Fact 5.6.1 *For all $\epsilon \leq 1/2$, there exists an algorithm requiring $(1 + O(\varepsilon)) \cdot OPT(\mathcal{I}_t) + O(\varepsilon^{-2})$ bins at all times t while using only $O(\varepsilon^{-1})$ amortized migration factor.*

This upper bound is trivial – it suffices to repack according to an AFPTAS whenever the volume changes by a multiplicative $(1+\epsilon)$ factor (for completeness we prove this fact in §C.3). The challenge here is in showing this algorithm's a.c.r to recourse tradeoff is *tight*. We do so by constructing an instance where addition or removal of small items of size $\approx \varepsilon$ causes *every* near-optimal solution to be far from every near-optimal solution after addition/removal.

For infinitely many $\epsilon > 0$, any fully-dynamic bin packing algorithm with a.c.r $(1+\varepsilon)$ and additive term $o(n)$ must have *amortized* migration factor of $\Omega(\varepsilon^{-1})$.

Our matching lower bound relies on the well-known Sylvester sequence [135], given by the recurrence relation $k_1 = 2$ and $k_{i+1} = \left(\prod_{j \leq i} k_j\right) + 1$, the first few terms of which are $2, 3, 7, 43, 1807, \ldots$ While this sequence has been used previously in the context of BIN PACKING, our proof relies on more fine-grained divisibility properties. In particular, letting c be a positive integer specified later and $\varepsilon := 1/\prod_{\ell=1}^{c} k_\ell$, we use the following properties:

(P1) $\frac{1}{k_1} + \frac{1}{k_2} + \ldots + \frac{1}{k_c} = 1 - \frac{1}{\prod_{\ell=1}^{c} k_\ell} = 1 - \varepsilon$.

(P2) If $i \neq j$, then k_i and k_j are relatively prime.

(P3) For all $i \in [c]$, the value $1/k_i = \prod_{\ell \in [c] \setminus \{i\}} k_\ell / \prod_{\ell=1}^{c} k_\ell$ is an integer product of ε.

(P4) If $i \neq j \in [c]$, then $1/k_i = \prod_{\ell \in [c] \setminus \{i\}} k_\ell / \prod_{\ell=1}^{c} k_\ell$ is an integer product of $k_j \cdot \epsilon$.

We define a vector of item sizes $\vec{s} \in [0,1]^{c+1}$ in our instances as follows: for $i \in [c]$ we let $s_i = \frac{1}{k_i} \cdot (1 - \frac{\varepsilon}{2})$, and $s_{c+1} = \varepsilon \cdot (\frac{3}{2} - \frac{\varepsilon}{2})$. The adversarial input sequence will alternate between two instances, $\mathcal{I}$ and $\mathcal{I}'$. For some large N a product of $\prod_{\ell=1}^{c} k_\ell$, Instance $\mathcal{I}$ consists of N items of sizes s_i for all $i \in [c+1]$. Instance $\mathcal{I}'$ consists of N items of all sizes but s_{c+1}.

Properties 1-4 imply on the one hand that $\mathcal{I}$ can be packed into completely full bins containing one item of each size, while any bin which does not contain exactly one item of each size has at least $\Omega(\varepsilon)$ free space. Similarly, an optimal packing of $\mathcal{I}'$ packs items of the same size in one set of bins, using up exactly $1 - \frac{\epsilon}{2}$ space, while any bin which contains items of at least two sizes has at least ϵ free space. These observations imply the following.

Any algorithm $\mathcal{A}$ with $(1+\varepsilon/7)$-a.c.r and $o(n)$ additive term packs instance $\mathcal{I}$ such that at least $2N/3$ bins contain exactly one item of each size s_i, and packs instance $\mathcal{I}'$ such that at least $N/2$ bins contain items of exactly one size.

Theorem 5.6 follows from Lemma 5.6 in a rather straightforward fashion. The full details of this proof and the lemmas leading up to it can be found in §C.3.

5.7 Open Problems

Subsequent to our work on this topic, one of the open questions was resolved by Feldkord et al. [61]. They showed that it is possible to achieve a a.c.r of ≈ 1.3871 using $O(\frac{1}{\varepsilon^2})$ worst-case recourse. There are two questions that remain in the general cost model.

Problem 5.7.1 *Does there exist a black-box reduction that converts any algorithm for the online bin-packing model into an algorithm achieving the same a.c.r in the fully dynamic binpacking model with recourse?*

The above work showed that this is true for the super-harmonic family. It remains to show this for the future algorithms that have many similar characteristics.

Problem 5.7.2 *Is there an algorithm which can a.c.r of* 1.58889 *with worst-case recourse* $O(\frac{1}{\varepsilon^2})$ *for the general cost model?*

Our algorithm showed this for jobs that are sufficiently large. It remains to show that this possible for small jobs.

Chapter 6

Online Matroid Intersection

6.1 Introduction

The *online matroid intersection problem* in the random arrival model (OMI) consists of two matroids $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$, where the elements in E are presented one-by-one to an online algorithm whose goal is to construct a large common independent set. As an element arrives, the algorithm must immediately and irrevocably decide whether to *pick* it, while ensuring that the picked elements always form a common independent set. We assume that the algorithm knows the size of E and has access to independence oracles for the already arrived elements. The greedy algorithm, which picks an element whenever possible, is half-competitive.

A special case of OMI where both the matroids are partition matroids already captures the *online bipartite matching problem* in the random edge arrival (OBME) model. Here, edges of a fixed (but adversarially chosen) bipartite graph G arrive in a uniformly random order and the algorithm must irrevocably decide whether to pick them into a matching. Despite tremendous progress made in the online vertex arrival model [1, 47, 68, 104, 110, 122, 138], nothing non-trivial was known in the edge arrival model where the edges arrive one-by-one. We present the first algorithm that performs better than greedy in the random arrival model. Besides being a natural theoretical question, it captures various online content systems such as online libraries where the participants are known to the matching agencies but the requests arrive in an online fashion.

More formally, OBME consists of a fixed bipartite graph G whose edges arrive one-by-one to an online algorithm in a uniformly random order. As an edge arrives, the algorithm must immediately and irrevocably decide whether to pick it into a matching. The algorithm knows the number of edges to arrive but must always maintain a matching. The objective is to maximize the size of the final matching.

Corollary 6.1.1 *The online bipartite matching problem in the random edge arrival model has a $(\frac{1}{2} + \delta)$-competitive randomized algorithm, where $\delta > 0$ is a constant.*

We also consider much more general problems of online matching in general graphs and online k-matroid intersection; the latter problem being NP-Hard.

6.2 Related Work

Our main OMI result is interesting in two different aspects: It gives the first linear time algorithm that beats greedy for the classical offline matroid intersection problem; also, it is the first non-trivial algorithm for the general problem of online matroid intersection, where previously nothing better than half was known even for online bipartite matching. Since offline matroid intersection problem is a fundamental problem in the field of combinatorial optimization [130, Chapter 41] and online matching occupies a central position in the field of online algorithms [120], there is a long list of work in both these areas. We state the most relevant works here and refer readers to further related work in the full paper.

Offline matroid intersection was brought to prominence in the groundbreaking work of Edmonds [51]. To illustrate the difficulty in moving from bipartite matching to matroid intersection, we note that while the first linear time algorithms that beat half for bipartite matching were designed more than 20 years old [11, 88], the fastest known matroid intersection algorithms that beat half make $\Omega(rm)$ calls to the independence oracles, where r is the rank of the optimal solution [37, 90]. The quadratic term appears because matroid intersection algorithms rely on constructing auxiliary graphs, which needs $\Omega(rm)$ calls [108, Chapter 13]. Until our work, it remained an open question to achieve a competitive ratio better than half with linear number of independence oracle calls. The key ingredient that allows us to circumvent these difficulties is the *Sampling Lemma* for matroid intersection. We do not construct an auxiliary graph and instead show that any maximal common independent is either already a $(\frac{1}{2} + \delta)$ approximation, or we can improve it to a $(\frac{1}{2} + \delta)$ approximation in a single pass over all the elements.

Online bipartite matching has been studied extensively in the vertex arrival model (see a nice survey by Mehta [120]). Since adversarial arrival order often becomes too pessimistic, the random arrival model (similar to the secretary problem) for online matching was first studied by Goel and Mehta [68]. Since then, this modeling assumption has become standard [99, 109, 110, 119]. The only progress when edges arrive one-by-one has been in showing lower bounds: no algorithm can achieve a competitive ratio better than 0.57 (see [53]), even when the algorithm may drop edges.

While nothing was previously known for online matching in the random edge arrival model, similar problems have been studied in the streaming model, most notably by Konrad et al. [107]. They gave the first algorithm that beats half for bipartite matching in the random arrival streaming model. In this work we generalize their *Hastiness Lemma* to matroids. However, prior works on online matching are not that useful to us as they are tailored to graphs—for instance their reliance on notion of "vertices" cannot be easily extended to the framework of matroids.

The simplicity of our OMI algorithm and flexibility of our analysis allows us to tackle problems of much greater generality, such as general graphs and k-matroid intersection, when previously even special cases like bipartite matching had been considered difficult in the online regime [121]. While our results are a qualitative advance, the quantitative improvement is small ($\delta > 10^{-4}$). It remains an interesting challenge to improve the approximation factor δ. Perhaps a more interesting challenge is to relax the random order requirement.

6.3 Our Results and Techniques

The following is the main result of this chapter.

Theorem 6.3.1 *The online matroid intersection problem in the random arrival model has a $(\frac{1}{2}+\delta)$-competitive randomized algorithm, where $\delta > 0$ is a constant.*

Our OMI algorithm makes only a linear number of calls to the independence oracles of both the matroids. Given recent interest in finding fast approximation algorithms for fundamental polynomial-time problems, this result is of independent interest even in the offline setting. Previously known algorithms that perform better than the greedy algorithm construct an "auxiliary graph", which already takes quadratic time [37, 90].

Corollary 6.3.2 *The matroid intersection problem has a linear time $(\frac{1}{2} + \delta)$ approximation algorithm, where $\delta > 0$ is a constant. .*

Finally, the simplicity of our OMI algorithm allows us to extend our results to the much more general problems of online matching in general graphs and to online k-matroid intersection; the latter problem being NP-Hard.

Theorem 6.3.3 *In the random edge arrival model, online matching problem for general graphs has a $(\frac{1}{2} + \delta')$-competitive randomized algorithm, where $\delta' > 0$ is a constant.*

Theorem 6.3.4 *The online k-matroid intersection problem in the random arrival model has a $\left(\frac{1}{k} + \frac{\delta''}{k^4}\right)$-competitive randomized algorithm, where $\delta'' > 0$ is a constant.*

In this section, we present an overview of our techniques to prove the main Theorem 6.3.1. Our analysis relies on two observations about the greedy algorithm that are encompassed in the Sampling Lemma and the Hastiness Lemma; former being the major contribution of this paper and the latter being useful to extend our linear time offline matroid intersection result to the online setting. Informally, the Sampling Lemma states that the greedy algorithm cannot perform poorly on a randomly generated OMI instance, and the hastiness lemma states that if the greedy algorithm performs poorly, then it picks most of its elements very quickly.

Let OPT denote a fixed maximum independent set in the intersection of matroids $\mathcal{M}_1$ and $\mathcal{M}_2$. WLOG, we assume that the greedy algorithm is *bad*—returns a common independent set T of size $\approx \frac{1}{2}|\text{OPT}|$. For offline matroid intersection, by running the greedy algorithm once, one can assume that T is known. For online matroid intersection, we use the Hastiness Lemma to construct T. It states that even if we run the greedy algorithm for a small fraction f (say $< 1\%$) of elements, it already picks a set T of elements of size $\approx \frac{1}{2}|\text{OPT}|$. This lemma was first observed by Konrad et al. [107] for bipartite matching and is generalized to matroid intersection in this work. By running the greedy algorithm for this small fraction f, the lemma lets us assume that we start with an approximately maximal common independent set T with most of the elements $(1 - f > 99\%)$ still to arrive.

The above discussion reduces the problem to improving a common independent set T of size $\approx \frac{1}{2}|\text{OPT}|$ to a common independent set of size $\geq (\frac{1}{2} + \delta)|\text{OPT}|$ in a single pass over all the elements. (This is true for both linear-time offline and OMI.) Since T is approximately maximal, we know that picking most elements in T eliminates the possibility of picking two OPT elements (one for each matroid). Hence, to beat half-competitiveness, we drop a uniformly random p fraction of these "bad" elements in T to obtain a set S, and try to pick $(1 + \gamma)$OPT elements (for constant

$\gamma > 0$) per dropped element. Our main challenge is to construct an online algorithm that can get on average γ gain per dropped element of T in a single pass. The Sampling Lemma for matroid intersection, which is our main technical contribution and forms the core of our analysis, comes to our rescue.

Sampling Lemma (informal): *Suppose T is a common independent set in matroids $\mathcal{M}_1$ and $\mathcal{M}_2$ and define $\tilde{E} = \mathsf{span}_1(T)$. Let S denote a random set containing each element of T independently with probability $(1-p)$. Then,*

$$\mathbb{E}_S[|Greedy(\mathcal{M}_1/S, \mathcal{M}_2/T, \tilde{E})|] \geq \left(\frac{1}{1+p}\right) \cdot \mathbb{E}_S[|\text{OPT}(\mathcal{M}_1/S, \mathcal{M}_2/T) \cap \tilde{E}|].$$

Intuitively, it says that for the random OMI instance on the contracted matroids $\mathcal{M}_1/S$ and $\mathcal{M}_2/T$, the expected size of greedy is more than $1/(1+p) \geq 1/2$ fraction of the optimal intersection. For $p < 1$, the advantage over half yields the γ gain per dropped element. The proof of the lemma involves giving an alternate view of the greedy algorithm for the random OMI instance. Using a carefully constructed invariant and the method of deferred decisions we show that the expected greedy solution is not too small.

Applying the Sampling Lemma requires a little care as we need to apply it twice, once for $(\mathcal{M}_1/S, \mathcal{M}_2/T)$ and once for $(\mathcal{M}_1/T, \mathcal{M}_2/S)$, while ensuring that the resulting solutions have few "conflicts" with each other. To overcome this, we only consider elements that are in the span of T for exactly one of the matroids.

6.4 Warmup: Online Bipartite Matching

In this section, we consider a special case of online matroid intersection, namely online bipartite matching in the random edge arrival model. Although, this is a special case of the general Theorem 6.3.1, we present it because nothing non-trivial was known before (see Section **??**) and several of our ideas greatly simplify in this case (in particular the Sampling Lemma), allowing us to lay the framework of our ideas.

6.4.1 Definitions and Notation

An instance of the online bipartite matching problem (G, E, π, m) consists of a bipartite graph $G = (U \cup V, E)$ with $m = |E|$, and where the edges in E arrive according to the order defined by π. We assume that the algorithm knows m but does not know E or π. For $1 \leq i \leq j \leq m$, let $E^\pi[i,j]$ denote the set of edges that arrive in between positions i through j according to π [1]. When permutation π is implicit, we abbreviate this to $E[i,j]$.

GREEDY denotes the algorithm that picks an edge into the matching whenever possible. Let OPT denote a fixed maximum offline matching of graph G. For $f \in [0,1]$, let T_f^π denote the matching produced by GREEDY after seeing the first f-fraction of the edges according to order π.

[1] We emphasize that our definition also works when i and j are non-integral

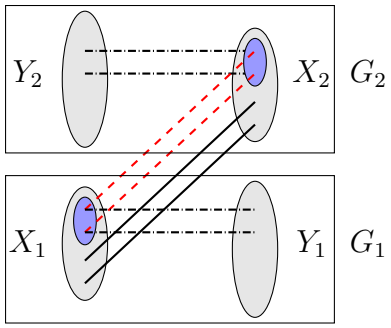

Figure 6.1: $U = X_1 \cup Y_2$ and $V = X_2 \cup Y_1$, where X_1 and X_2 denote the set of vertices matched by GREEDY in Phase (a). Here thick-edges are picked and diagonal-dashed-edges are marked. Horizontal-dashed-edges show augmentations for the marked edges.

For a uniformly random chosen order π,

$$\mathcal{G}(f) := \frac{\mathbb{E}_\pi[|T_f^\pi|]}{|\text{OPT}|}.$$

Hence, $\mathcal{G}(1)\,|\text{OPT}|$ is the expected output size of GREEDY and $\mathcal{G}(\frac{1}{2})\,|\text{OPT}|$ is the expected output size of GREEDY after seeing half of the edges. We observe that GREEDY has a competitive ratio of at-least half and in Section B.2, we show that this ratio is tight for worst case input graphs.[2]

6.4.2 Beating Half

Lemma 6.4.1 shows that we can restrict our attention to the case when the expected GREEDY size is small. Theorem 6.4.2 gives an algorithm that beats half for this restricted case.

Lemma 6.4.1 *Suppose there exists an Algorithm $\mathcal{A}$ that achieves a competitive ratio of $\frac{1}{2}+\gamma$ when $\mathcal{G}(1) \leq (\frac{1}{2}+\epsilon)$ for some $\epsilon, \gamma > 0$. Then there exists an algorithm with competitive ratio at least $\frac{1}{2}+\delta$, where $\delta = \frac{\epsilon\gamma}{\frac{1}{2}+\epsilon+\gamma}$.*

Proof: Consider the algorithm that tosses a coin at the beginning and runs GREEDY with probability $1-r$ and Algorithm $\mathcal{A}$ with probability r, where $r > 0$ is some constant. This lemma follows from simple case analysis.

- Case 1: $\mathcal{G}(1) < \frac{1}{2}+\epsilon$
  Since GREEDY is always $\frac{1}{2}$ competitive, we can say that in expectation, the competitive ratio will be at least
  $$(1-r)\,\frac{1}{2} + r\left(\frac{1}{2}+\gamma\right) = \frac{1}{2} + r\gamma$$
- Case 2: $\mathcal{G}(1) \geq \frac{1}{2}+\epsilon$
  Since we have no guarantees on the performance of Algorithm $\mathcal{A}$ when GREEDY performs

[2]We also show that for regular graphs the competitive ratio of GREEDY is at least $\left(1-\frac{1}{e}\right)$, and that no online algorithm for OBME can give a ratio better than $\frac{69}{84} \approx 0.821$.

well, we assume that it achieves a competitive ratio of 0. Our expected performance will be at least

$$(1-r)\left(\frac{1}{2}+\epsilon\right)+0=\frac{1}{2}+\epsilon-\frac{r}{2}-r\epsilon$$

Choosing $r=\frac{\epsilon}{\frac{1}{2}+\epsilon+\gamma}$, we get $\delta \geq \frac{\epsilon\gamma}{\frac{1}{2}+\epsilon+\gamma}$. ■

Theorem 6.4.2 *If $\mathcal{G}(1) \leq (\frac{1}{2}+\epsilon)$ for some constant $\epsilon > 0$ then the* MARKING-GREEDY *algorithm outputs a matching of size at least $(\frac{1}{2}+\gamma)\,|\mathrm{OPT}|$ in expectation, where $\gamma > 0$ is a constant.*

Before describing MARKING-GREEDY, we need the following property about the performance of GREEDY in the random arrival model — if GREEDY is bad then it makes most of its decisions quickly and incorrectly. We will be interested in the regime where $0 < \epsilon \ll f \ll 1$.

Lemma 6.4.3 (Hastiness property: Lemma 2 in [107]) *For any graph G if $\mathcal{G}(1) \leq (\frac{1}{2}+\epsilon)$ for some $0 < \epsilon < \frac{1}{2}$, then for any $0 < f < 1/2$*

$$\mathcal{G}(f) \geq \frac{1}{2} - \left(\frac{1}{f}-2\right)\epsilon.$$

MARKING-GREEDY **for bipartite matching:**

MARKING-GREEDY consists of two phases (see the pseudocode). In Phase (a), it runs GREEDY for the first f-fraction of the edges, but *picks* each edge *selected* by GREEDY into the final matching only with probability $(1-p)$, where $p > 0$ is a constant. With the remaining probability p, it *marks* the edge e and its vertices, and behaves as if it had been picked. In Phase (b), which is for the remaining $1-f$ fraction of edges, the algorithm runs GREEDY to pick edges on two restricted disjoint subgraphs G_1 and G_2, where it only considers edges incident to exactly one marked vertex in Phase (a). (see Figure 6.1.)

Phase (a) is equivalent to running GREEDY to select elements, but then randomly dropping p fraction of the selected edges. The idea of marking some vertices (by marking an incident edge) is to "protect" them for augmentation in Phase (b). To distinguish if an edge is marked or picked, the algorithm uses auxiliary random bits Ψ that are unknown to the adversary. We assume that $\Psi(e) \sim \mathrm{Bern}(1-p)$ i.i.d. for all $e \in E$.

Comparison to Konrad et al. [107] For the special case of bipartite matching, we can consider MARKING-GREEDY to be a variant of the streaming algorithm of [107]. For graphs where GREEDY is bad, both algorithms use the first phase to pick an approximately maximal matching T using the Hastiness Lemma. [107] divides the remaining stream into two portions and uses each portion to find greedy matchings, say F_1 and F_2. Since decisions in the streaming setting are revocable, at the end of the stream they use edges in $F_1 \cup F_2$ to find sufficient number of three-augmenting paths w.r.t. T. Their algorithm is not online because it keeps all the matchings till the end. One can view the current algorithm as turning their algorithm into an online one by flipping a coin for each edge in T. In the second phase, it runs GREEDY on two random disjoint subgraphs and use the Sampling Lemma to argue that in expectation the algorithm picks sufficient number of augmenting paths.

While our online matching algorithm is simple and succinct, the main difficulty lies in extending it to OMI as the notions of marking and protecting vertices do not exist. This is also the reason

Algorithm MARKING-GREEDY(G, E, π, m, Ψ)

Phase (a)

Initialize S, T, N_1, N_2 to $\emptyset$
for each element $e \in E^\pi[1, fm]$ **do** ▷ GREEDY *while picking and marking*
  if $T \cup e$ is a matching in G **then**
    $T \leftarrow T \cup e$ ▷ *Elements selected by* GREEDY
    if $\Psi(e) = 1$ **then** ▷ *Auxiliary random bits* Ψ
      $S \leftarrow S \cup e$ ▷ *Elements picked into final solution*

Phase (b)

Initialize set T_f to T. Let sets X_1, X_2 be vertices of U, V matched in T_f respectively.
Let G_1 be the subgraph of G induced on X_1 and $V \setminus X_2$.
Let G_2 be the subgraph of G induced on $U \setminus X_1$ and X_2.
for each edge $e \in (E^\pi[fm, m])$ **do** ▷ GREEDY *on two disjoint subgraphs*
  for $i \in \{1, 2\}$ **do**
    if $e \in G_i$ and $S \cup N_i \cup e$ is a matching **then** ▷ Greedy step
      $N_i \leftarrow N_i \cup e$ ▷ *New edges picked*
return $S \cup N_1 \cup N_2$

why obtaining a linear time algorithm for offline matroid intersection problem, where Hastiness Lemma is not needed, had been open. Defining and proving the correct form of Sampling Lemma forms the core of our OMI analysis in Section 6.5.

Proof that MARKING-GREEDY **works for bipartite matching:**

Let G_i denote graphs G_1 or G_2 for $i \in \{1, 2\}$. For a fixed order π of the edges, graphs G_i in MARKING-GREEDY are independent of the randomness Ψ. Since the algorithm uses Ψ to pick a random subset of the GREEDY solution, this can be viewed as independently sampling each vertex matched by GREEDY in G_i. Lemma 6.4.4 shows that this suffices to pick in expectation more than the number of marked edges. In essence, we use the randomness Ψ to limit the power of an adversary deciding the order of the edges in Phase (b). While the proof follows from the more general Lemma 6.5.7, we include a simple self-contained proof.

Lemma 6.4.4 (Sampling Lemma) *Consider a bipartite graph $H = (X \cup Y, \tilde{E})$ containing a matching $\tilde{I}$. Let $\Psi(x) \sim$ Bern$(1-p)$ i.i.d. for all $x \in X$, and define $X' = \{x \mid x \in X$ and $\Psi(x) = 0\}$. I.e., the vertices of X' are obtained by independently sampling each vertex in X with probability p. Let H' denote the subgraph induced on X' and Y. Then for any arrival order of the edges in H',*

$$\mathbb{E}_\Psi[\text{GREEDY}(H', \tilde{E})] \geq \frac{p}{1+p}|\tilde{I}|.$$

Proof: We prove this statement by induction on $|\tilde{I}|$. Consider the base case $|\tilde{I}| = 1$. Whenever GREEDY does not select any edge, the vertex adjacent to $\tilde{I}$ in X is not sampled. This happens with

probability $1-p$. Hence, the expected size of the matching is at least $p \geq \frac{p}{1+p}$, which implies the statement is true when $|\tilde{I}| = 1$.

From the induction hypothesis (I.H.) we can assume the statement is true when the matching size is at most $|\tilde{I}|-1$. We prove the induction step by contradiction and consider the smallest graph in terms of $|X|$ that does not satisfy the statement. Note that $|X| \geq |\tilde{I}|$. Consider the first edge $e = (x, y)$ that arrives. The first case is when $x \notin X'$ and it happens with probability $1-p$. Here any edge incident to x does not matter for the remaining algorithm. We use I.H. on the subgraph induced on $(X \backslash x, Y)$ as $|X \backslash x| = (|X| - 1)$. Since this subgraph has a matching of size at least $|\tilde{I}| - 1$, I.H. gives a matching of expected size at least $\frac{p}{1+p}(|\tilde{I}| - 1)$.

The second case is when $x \in X'$ and it happens with probability p. Now edge (x, y) is included in the GREEDY matching for the induced graph on (X', Y). Vertices x and y, along with the edges incident to them, do not participate in the remaining algorithm. We apply I.H. on the subgraph induced on the vertices $(X \backslash x, Y \backslash y)$. Noting that this graph has a matching of size at least $|\tilde{I}| - 2$, I.H. gives a matching of expected size at least $\frac{p}{1+p}(|\tilde{I}| - 2)$. Combining both cases, the expected matching size is at least

$$(1-p)\left(\frac{p}{1+p}(|\tilde{I}|-1)\right) + p\left(1 + \frac{p}{1+p}(|\tilde{I}|-2)\right) = \frac{p}{1+p}|\tilde{I}|.$$

This is a contradiction as we assumed that the graph did not satisfy the induction statement, which completes the proof of Lemma 6.4.4. ■

We next prove the main lemma needed to prove Theorem 6.4.2. Setting $f = 0.07$, $p = 0.36$, and $\epsilon = 0.001$ in Lemma 6.4.5, the theorem immediately follows with $\gamma > 0.05$.

Lemma 6.4.5 *For any $0 < f < 1/2$ and bipartite graph G,* MARKING-GREEDY *outputs a matching of expected size at least*

$$\left[(1-p)\left(\frac{1}{2} - \left(\frac{1}{f} - 2\right)\epsilon\right) + \frac{p}{1+p}\left(1 - \frac{2\epsilon}{f} - f\right)\right]|\text{OPT}|.$$

Proof: We remind the reader that for any $f \in [0, 1]$ and any permutation π of the edges, T_f^π denotes the matching that GREEDY produces on $E^\pi[1, fm]$. For $i \in \{1, 2\}$, let H_i denote the subgraph of G_i containing all its edges that appear in Phase (b). Let I_i denote the set of edges of OPT that appear in graph G_i. We use the following claim.

Claim 6.4.6

$$\mathbb{E}_\pi\left[|I_1| + |I_2|\right] \geq \left(1 - \frac{2\epsilon}{f}\right)|\text{OPT}|.$$

Proof: We use the following two simple properties of T_1^π (proved in Section B.3).

Fact 6.4.7

$$|T_1^\pi| \geq \frac{1}{2}\left(|\text{OPT}| + \sum_{e \in \text{OPT}} \mathbf{1}[\textit{Both ends of } e \textit{ matched in } T_f^\pi]\right) \textit{ and} \tag{6.1}$$

$$|T_1^\pi| \geq |T_f^\pi| + \frac{1}{2}\sum_{e \in \text{OPT}} \mathbf{1}[\textit{Both ends of } e \textit{ unmatched in } T_f^\pi]. \tag{6.2}$$

Note that, $\mathbb{E}_\pi\left[|I_1| + |I_2|\right]$ is equal to

$$\begin{aligned}
&\mathbb{E}_\pi\left[|\text{OPT}| - \sum_{e\in\text{OPT}} \mathbf{1}[\text{Both ends of } e \text{ matched in } T_f^\pi] - \sum_{e\in\text{OPT}} \mathbf{1}[\text{Both ends of } e \text{ unmatched in } T_f^\pi]\right] \\
&\geq\ |\text{OPT}| - \mathbb{E}_\pi\left[2\,|T_1^\pi| - |\text{OPT}|\right] - \mathbb{E}_\pi\left[2(|T_1^\pi| - |T_f^\pi|)\right] \qquad \text{(using Eq. (6.1) and Eq. (6.2))} \\
&\geq\ |\text{OPT}| - 2\epsilon\,|\text{OPT}| - 2\left(\epsilon + \left(\frac{1}{f} - 2\right)\epsilon\right)|\text{OPT}| \qquad \text{(using } \mathcal{G}(1) \leq \tfrac{1}{2} + \epsilon \text{ and Lemma 6.4.3)} \\
&=\ \left(1 - \frac{2\epsilon}{f}\right)|\text{OPT}|, \text{ which finishes the proof of the claim.}
\end{aligned}$$

■

For $i \in \{1, 2\}$, let $\tilde{I}_i \subseteq I_i$ denote the set of edges of OPT that appear in Phase (b) of MARKING-GREEDY, i.e., they appear in graph H_i. In expectation over uniform permutation π, at most $f|\text{OPT}|$ elements of OPT can appear in Phase (a). Hence,

$$\mathbb{E}_\pi\left[|\tilde{I}_1| + |\tilde{I}_2|\right] \geq \mathbb{E}_\pi\left[|I_1| + |I_2|\right] - f|\text{OPT}| \geq \left(1 - \frac{2\epsilon}{f} - f\right)|\text{OPT}|.$$

Marking a random subset of T_f^π independently is equivalent to marking a random subset of vertices independently. Thus, we can apply Lemma 6.4.4 to both H_1 and H_2. The expected number of edges in $N_1 \cup N_2$ is at least $\frac{p}{1+p}(|\tilde{I}_1| + |\tilde{I}_2|)$, where the expectation is over the auxilary bits Ψ that distinguishes the random set of edges marked. Taking expectations over π and noting that Phase (a) picks $(1-p)\,\mathcal{G}(f)\,|\text{OPT}|$ edges, we have

$$\begin{aligned}
\mathbb{E}_{\Psi,\pi}[|S \cup N_1 \cup N_2|] \quad &= \quad \mathbb{E}_{\Psi,\pi}[|S|] + \mathbb{E}_{\Psi,\pi}[|N_1| + |N_2|] \\
&\geq\ \mathcal{G}(f)(1-p)\,|\text{OPT}| + \frac{p}{1+p}\mathbb{E}_\pi\left[|\tilde{I}_1| + |\tilde{I}_2|\right] \\
&\geq\ \left[(1-p)\left(\frac{1}{2} - \left(\frac{1}{f} - 2\right)\epsilon\right) + \frac{p}{1+p}\left(1 - \frac{2\epsilon}{f} - f\right)\right]|\text{OPT}| \quad \text{(using Lemma 6.4.3)}.
\end{aligned}$$

■

6.5 Online Matroid Intersection

6.5.1 Definitions and Notation

An instance of the online matroid intersection problem $(\mathcal{M}_1, \mathcal{M}_2, E, \pi, m)$ consists of matroids $\mathcal{M}_1$ and $\mathcal{M}_2$ defined on ground set E of size m, and where the elements in E arrive according to the order defined by π. For any $1 \leq i \leq j \leq m$, let $E^\pi[i, j]$ denote the ordered set of elements of E that arrive in positions i through j according to π. For any matroid $\mathcal{M}$ on ground set E, we use $T \in \mathcal{M}$ to denote $T \subseteq E$ is an independent set in matroid $\mathcal{M}$. We use the terminology of matroid restriction and matroid contraction as defined in Oxley [126]. To avoid clutter, for any $e \in E$ we will abbreviate $A \cup \{e\}$ to $A \cup e$ and $A \setminus \{e\}$ to $A \setminus e$.

Algorithm GREEDY $(\mathcal{M}_1, \mathcal{M}_2, E, \pi)$

Initialize set T to $\emptyset$
for each element $e \in E^\pi[1, |E|]$ **do**
    if $T \cup e \in \mathcal{M}_1 \cap \mathcal{M}_2$ **then**
        $T \leftarrow T \cup e$
return T

We note that GREEDY is well defined even when matroids $\mathcal{M}_1$ and $\mathcal{M}_2$ are defined on larger ground sets as long as they contain E. This notation will be useful when we run GREEDY on matroids after contracting different sets in the two matroids. Since GREEDY always produces a maximal independent set, it has a competitive ratio of at least half (see Theorem 13.8 in [108]). This is true because addition of an "incorrect" element to OPT can create at most two circuits, one for each matroid.

Let OPT denote a fixed maximum offline independent set in the intersection of both the matroids. For $f \in [0, 1]$, let T_f^π denote the independent set that GREEDY produces after seeing the first f fraction of the edges according to order π. When clear from context, we will often abbreviate T_f^π with T_f. Let $\mathcal{G}(f) := \frac{\mathbb{E}_\pi[|T_f|]}{|\text{OPT}|}$, where π is a uniformly random chosen order.

For $i \in \{1, 2\}$, let $\mathsf{span}_i(T) := \{e \mid (e \in E) \wedge (\mathsf{rank}_{\mathcal{M}_i}(T \cup e) = \mathsf{rank}_{\mathcal{M}_i}(T))\}$ denote the span of set $T \subseteq E$ in matroid $\mathcal{M}_i$. Suppose we have $T \in \mathcal{M}_i$ and $e \in \mathsf{span}_i(T)$, then we denote the unique circuit of $T \cup e$ in matroid $\mathcal{M}_i$ by $C_i(T \cup e)$. If $i = 1$, we use $\bar{\imath}$ to denote 2, and vice versa.

We provide a table of all notation used in Section B.1.

6.5.2 Hastiness Property

Before describing our algorithm MARKING-GREEDY, we need an important hastiness property of GREEDY in the random arrival model. Intuitively, it states that if GREEDY's performance is bad then it makes most of its decisions quickly and incorrectly. This observation was first made by Konrad et al. [107] in the special case of bipartite matching. We extend this property to matroids in Lemma 6.5.1 (proof in Section B.4). We are interested in the regime where $0 < \epsilon \ll f \ll 1$.

Lemma 6.5.1 (Hastiness Lemma) *For any two matroids $\mathcal{M}_1$ and $\mathcal{M}_2$ on the same ground set E, let T_f^π denote the set selected by* GREEDY *after running for the first f fraction of elements E appearing in order π. Also, for $i \in \{1, 2\}$, let $\Phi_i(T_f^\pi) := \mathsf{span}_i(T_f^\pi) \cap \text{OPT}$. Now for any $0 < f, \epsilon \leq \frac{1}{2}$, if $\mathbb{E}_\pi[|T_1^\pi|] \leq (\frac{1}{2} + \epsilon)\,|\text{OPT}|$ then*

$$\mathbb{E}_\pi\left[|\Phi_1(T_f^\pi) \cap \Phi_2(T_f^\pi)|\right] \leq 2\epsilon\,|\text{OPT}| \qquad \textit{and}$$
$$\mathbb{E}_\pi\left[|\Phi_1(T_f^\pi) \cup \Phi_2(T_f^\pi)|\right] \geq \left(1 - \frac{2\epsilon}{f} + 2\epsilon\right)|\text{OPT}|.$$

This implies $\mathcal{G}(f) := \frac{\mathbb{E}_\pi[|T_f^\pi|]}{|\text{OPT}|} \geq \left(\frac{1}{2} - \left(\frac{1}{f} - 2\right)\epsilon\right)$.

6.5.3 Beating Half for Online Matroid Intersection

Once again, we use Lemma 6.4.1 to restrict our attention to the case when the expected size of GREEDY is small. In Theorem 6.5.2, we give an algorithm that beats half for this restricted case, which when combined with Lemma 6.4.1 finishes the proof of Theorem 6.3.1.

Theorem 6.5.2 *For any two matroids $\mathcal{M}_1$ and $\mathcal{M}_2$ on the same ground set E, there exist constants $\epsilon, \gamma > 0$ and a randomized online algorithm* MARKING-GREEDY *such that if $\mathcal{G}(1) \leq \left(\frac{1}{2} + \epsilon\right)$ then* MARKING-GREEDY *outputs an independent set in the intersection of both the matroids of expected size at least $\left(\frac{1}{2} + \gamma\right)|\text{OPT}|$.*

MARKING-GREEDY **for OMI:**

Algorithm MARKING-GREEDY $(\mathcal{M}_1, \mathcal{M}_2, E, \pi, m, \Psi)$

Phase (a)
Initialize S, T to $\emptyset$
for each element $e \in E^{\pi}[1, fm]$ **do** ▷ GREEDY *while picking and marking*
  if $T \cup e \in \mathcal{M}_1 \cap \mathcal{M}_2$ **then**
    $T \leftarrow T \cup e$ ▷ *Elements selected by* GREEDY
    if $\psi(e) = 1$ **then** ▷ *Auxiliary random bits* Ψ
      $S \leftarrow S \cup e$ ▷ *Elements picked into the final solution*
Phase (b)
Fix T_f to T and initialize sets N_1, N_2 to $\emptyset$
for each element $e \in E^{\pi}[fm, m]$ **do** ▷ GREEDY *on two disjoint problems*
  for $i \in \{1, 2\}$ **do**
    if $e \in \mathsf{span}_i(T_f)$ and $e \notin \mathsf{span}_{\bar{i}}(T_f)$ **then** ▷ *To ensure disjointness*
      if $(S \cup N_i \cup e \in \mathcal{M}_i)$ and $(T_f \cup N_i \cup e \in \mathcal{M}_{\bar{i}})$ **then** ▷ Greedy step
        $N_i \leftarrow N_i \cup e$ ▷ *Newly picked elements*
return $(S \cup N_1 \cup N_2)$

MARKING-GREEDY consists of two phases (see notation in Section B.1). In Phase (a), it runs GREEDY for the first f fraction of the elements, but *picks* each element *selected* by GREEDY into the final solution only with probability $(1 - p)$, where $p > 0$ is a constant. With the remaining probability p, it *marks* the element e, and behaves as if it had been selected. The idea of marking some elements in Phase (a) is that we hope to "augment" them in Phase (b). To distinguish if an element is marked or picked, the algorithm uses auxiliary random bits Ψ that are unknown to the adversary. We assume that $\Psi(e) \sim \text{Bern}(1 - p)$ i.i.d. for all $e \in E$.

In Phase (b), one needs to ensure that the augmentations of the marked elements do not conflict with each other. The crucial idea is to use the span of the elements selected by GREEDY in Phase (a) as a proxy to find two random disjoint OMI subproblems. The following Fact 6.5.3 (proof in Section B.3) underlies this intuition. It states that given any independent set S, we can substitute

it by any other independent set contained in the span of S. In Lemma 6.5.4 we use it to prove the correctness of MARKING-GREEDY.

Fact 6.5.3 *Consider any matroid $\mathcal{M}$ and independent sets $A, B, C \in \mathcal{M}$ such that $A \subseteq \mathsf{span}_{\mathcal{M}}(B)$ and $B \cup C \in \mathcal{M}$. Then, $A \cup C \in \mathcal{M}$.*

Lemma 6.5.4 MARKING-GREEDY *outputs sets S, N_1, and N_2 such that*

$$(S \cup N_1 \cup N_2) \in \mathcal{M}_1 \cap \mathcal{M}_2.$$

Proof: Observe that the outputs sets S, N_1, and N_2 of MARKING-GREEDY satisfy the following for $i \in \{1, 2\}$:

$$N_i \in \mathcal{M}_i/S \cap \mathcal{M}_{\bar{i}}/T_f \qquad \text{(due to Line 11)} \tag{6.3}$$
$$N_i \subseteq \mathsf{span}_{\mathcal{M}_i/S}(T_f \setminus S) \qquad \text{(due to Line 10)} \tag{6.4}$$

From Property (6.3) above we know $N_{\bar{i}} \in \mathcal{M}_i/T_f$, which implies $N_{\bar{i}} \cup (T_f \setminus S) \in \mathcal{M}_i/S$ because $S \subseteq T_f \in \mathcal{M}_i$. Also, Property (6.4) implies $N_i \subseteq \mathsf{span}_{\mathcal{M}_i/S}(T_f \setminus S)$. Using Fact 6.5.3, we have $\big(N_1 \cup N_2\big) \in \mathcal{M}_i/S$. ■

Proof that MARKING-GREEDY **works for OMI:**

We know from Lemma 6.5.1 that $\mathcal{G}(f)$ is close to half for $\epsilon \ll f \ll 1$. In the following Lemma 6.5.5, we show that MARKING-GREEDY (which returns $S \cup N_1 \cup N_2$ by Lemma 6.5.4) gets an improvement over GREEDY. This completes the proof of Theorem 6.5.2 to give $\gamma \geq 0.03$ for $\epsilon = 0.001$, $f = 0.05$, and $p = 0.33$. The rest of the section is devoted to proving the following lemma.

Lemma 6.5.5 MARKING-GREEDY *outputs sets S, N_1, and N_2 such that*

$$\mathbb{E}_{\pi,\Psi}[|S \cup N_1 \cup N_2|] \geq (1-p)\,\mathcal{G}(f)\,|\mathbf{OPT}| + \frac{2p}{1+p}\left(1 - \frac{2\epsilon}{f} - 2\epsilon - f - \mathcal{G}(f)\right)|\mathbf{OPT}|.$$

Proof:[Lemma 6.5.5] We treat the sets $S \subseteq T_f, N_1$, and N_2 as random sets depending on π and Ψ. Since MARKING-GREEDY ensures the sets are disjoint,

$$\begin{aligned}\mathbb{E}_{\pi,\Psi}[|S \cup N_1 \cup N_2|] &= \mathbb{E}_{\pi,\Psi}[|S|] + \mathbb{E}_{\pi,\Psi}[|N_1| + |N_2|] \\ &\geq (1-p)\,\mathcal{G}(f)\,|\mathbf{OPT}| + \mathbb{E}_{\pi,\Psi}[|N_1| + |N_2|]. \end{aligned} \tag{6.5}$$

Next, we lower bound $\mathbb{E}_{\pi,\Psi}[|N_1| + |N_2|]$ by observing that for $i \in \{1, 2\}$, N_i is the result of running GREEDY on the following restricted set of elements.

Definition 6.5.6 (Sets $\tilde{E}_i$) *For $i \in \{1, 2\}$, we define $\tilde{E}_i$ to be the set of elements e that arrive in Phase (b) and satisfy $e \in \mathsf{span}_i(T_f)$ and $e \notin \mathsf{span}_{\bar{i}}(T_f)$.*

It's easy to see that N_i is obtained by running GREEDY on the matroids $\mathcal{M}_i/S$ and $\mathcal{M}_{\bar{i}}/T_f$ with respect to elements in $\tilde{E}_i$, i.e. $N_i = \text{GREEDY}(\mathcal{M}_i/S, \mathcal{M}_{\bar{i}}/T_f, \tilde{E}_i)$. To lower bound $\mathbb{E}_{\pi,\Psi}[|N_1| + |N_2|]$, we use the following Sampling Lemma (proved in Section 6.6) that forms the core of our technical analysis. Intuitively, it says that if S is a random subset of T_f then for the obtained random OMI instance, with optimal solution of expected size $p\,|\tilde{I}|$, GREEDY performs better than half-competitiveness even for adversarial arrival order of ground elements.

Lemma 6.5.7 (Sampling Lemma) *Given matroids $\mathcal{M}_1, \mathcal{M}_2$ on ground set E, a set $T \in \mathcal{M}_1 \cap \mathcal{M}_2$, and $\Psi(e) \sim$ Bern$(1-p)$ i.i.d. for all $e \in T$, we define set $S := \{e \mid e \in T \text{ and } \Psi(e) = 1\}$. I.e., S is a set achieved by dropping each element in T independently with probability p. For $i \in \{1,2\}$, consider a set $\tilde{E} \subseteq \mathsf{span}_i(T)$ and a set $\tilde{I} \subseteq \tilde{E}$ satisfying $\tilde{I} \in \mathcal{M}_i \cap (\mathcal{M}_{\bar{i}}/T)$. Then for any arrival order of the elements of $\tilde{E}$, we have*

$$\mathbb{E}_{\Psi}[\text{GREEDY}(\mathcal{M}_i/S, \mathcal{M}_{\bar{i}}/T, \tilde{E})] \geq \frac{1}{1+p}\left(p\,|\tilde{I}|\right).$$

To use the Sampling Lemma, in Claim 6.5.8 we argue that in expectation there exist disjoint sets $\tilde{I}_i \subseteq \tilde{E}_i$ of "large" size that satisfy the preconditions of the Sampling Lemma (proof uses Hastiness Lemma and is deferred to Section **??**).

Claim 6.5.8 *If $\mathcal{G}(1) \leq \left(\frac{1}{2} + \epsilon\right)$ then for $i \in \{1,2\}$ there exist disjoint sets $\tilde{I}_i \subseteq \tilde{E}_i$ s.t.*

(i) $\mathbb{E}_{\pi}\left[|\tilde{I}_1| + |\tilde{I}_2|\right] \geq 2\left(1 - \frac{2\epsilon}{f} - f - \mathcal{G}(f)\right)|\text{OPT}|$.

(ii) $\tilde{I}_i \in \mathcal{M}_i \cap (\mathcal{M}_{\bar{i}}/T_f)$.

Finally, to finish the proof of Lemma 6.5.5, we use the sets $\tilde{I}_i$ from the above Claim 6.5.8 as $\tilde{I}$ and sets $\tilde{E}_i$ as $\tilde{E}$ in the Sampling Lemma 6.5.7. From Eq. (6.5) and Claim 6.5.8, we get

$$\begin{aligned}\mathbb{E}_{\pi,\Psi}[|S \cup N_1 \cup N_2|] &\geq (1-p)\,\mathcal{G}(f)\,|\text{OPT}| + \frac{p}{1+p}\,\mathbb{E}_{\pi}\left[|\tilde{I}_1| + |\tilde{I}_2|\right] \\ &\geq (1-p)\,\mathcal{G}(f)\,|\text{OPT}| + \frac{2p}{1+p}\left(1 - \frac{2\epsilon}{f} - f - \mathcal{G}(f)\right)|\text{OPT}|.\end{aligned}$$

■

Proof:[Claim 6.5.8] Recall $\Phi_i(T_f^{\pi}) := \mathsf{span}_i(T_f^{\pi}) \cap \text{OPT}$. Let I_i denote $\Phi_i(T_f^{\pi}) \setminus \Phi_{\bar{i}}(T_f^{\pi})$. We construct sets $\tilde{I}_i$ by removing some elements from I_i, which implies $\tilde{I}_i \in \mathcal{M}_i$ because $I_i \in \mathcal{M}_i$. We first show that $|I_1| + |I_2|$ is large. From the Hastiness Lemma 6.5.1, we have

$$\begin{aligned}\mathbb{E}_{\pi}\left[|I_1| + |I_2|\right] &= \mathbb{E}_{\pi}\left[|\Phi_1(T_f^{\pi}) \cup \Phi_2(T_f^{\pi})|\right] - \mathbb{E}_{\pi}\left[|\Phi_1(T_f^{\pi}) \cap \Phi_2(T_f^{\pi})|\right] \\ &\geq \left(1 - \frac{2\epsilon}{f}\right)|\text{OPT}|. \qquad (6.6)\end{aligned}$$

Next, we ensure that $\tilde{I}_i \in \mathcal{M}_{\bar{i}}/T_f$. Note that $I_{\bar{i}} \subseteq \mathsf{span}_{\bar{i}}(T_f)$. Let $X_{\bar{i}}$ denote a minimum subset of elements of T_f such that $\mathsf{span}_{\bar{i}}(X_{\bar{i}} \cup I_{\bar{i}}) = \mathsf{span}_{\bar{i}}(T_f)$. Since $I_{\bar{i}}$ and T_f are independent in $\mathcal{M}_{\bar{i}}$, we have $|X_{\bar{i}}| = |T_f| - |I_{\bar{i}}|$. Now starting with $(I_i \cup I_{\bar{i}}) \in \mathcal{M}_{\bar{i}}$, we add elements of $X_{\bar{i}}$ into it. We will remove at most $|X_{\bar{i}}|$ elements from I_i to get a set I_i' such that $(I_i' \cup X_{\bar{i}} \cup I_{\bar{i}}) \in \mathcal{M}_{\bar{i}}$ as $(X_{\bar{i}} \cup I_{\bar{i}}) \in \mathcal{M}_{\bar{i}}$.

Using Fact 6.5.3 and $\mathsf{span}_{\bar{i}}(X_{\bar{i}} \cup I_{\bar{i}}) = \mathsf{span}_{\bar{i}}(T_f)$, we also have $I'_i \cup T_f \in \mathcal{M}_{\bar{i}}$. One can use a similar argument to obtain set $I'_{\bar{i}}$ and X_i such that $I'_{\bar{i}} \cup T_f \in \mathcal{M}_i$. Since $\mathbb{E}_\pi\big[|X_i|\big] = \mathbb{E}_\pi\big[|T_f| - |I_i|\big]$,

$$\mathbb{E}_\pi[|I'_1| + |I'_2|] \geq \mathbb{E}_\pi[|I_1| + |I_2| - |X_1| - |X_2|] = 2\,\mathbb{E}_\pi[|I_1| + |I_2| - |T_f|] \tag{6.7}$$

Finally, to ensure that $\tilde{I}_i \subseteq \tilde{E}_i$, observe that any element $e \in I'_i$ already satisfies $e \in \mathsf{span}_i(T_f)$ and $e \notin \mathsf{span}_{\bar{i}}(T_f)$. To ensure that these elements also appear in Phase (b), note that all elements of I'_i belong to OPT. Hence, in expectation over π, at most $f\,|\text{OPT}|$ of these elements can appear in Phase (a). The remaining elements appear in Phase (b). Thus, combining the following equation with Eq. (6.6) and Eq. (6.7) completes the proof of Lemma 6.5.5

$$\mathbb{E}_\pi\big[|\tilde{I}_1| + |\tilde{I}_2|\big] \geq \mathbb{E}_\pi\big[|I'_1| + |I'_2|\big] - f\,|\text{OPT}|.$$

■

6.6 Sampling Lemma

We prove the lemma for $i = 1$ as the other case is analogous.

6.6.1 Alternate View of the Sampling Lemma

We prove the Sampling Lemma by first showing that GREEDY$(\mathcal{M}_1/S, \mathcal{M}_2/T, \tilde{E})$ produces the same output as algorithm SAMP-ALG (proof deferred to Section 6.6.3).

Lemma 6.6.1 *Given a fixed Ψ and assuming the elements of $\tilde{E}$ are presented in the same order, the output of* SAMP-ALG *is the same as the output of* GREEDY$(\mathcal{M}_1/S, \mathcal{M}_2/T, \tilde{E})$.

The idea behind SAMP-ALG is to run GREEDY, but postpone distinguishing between the elements that are selected by GREEDY (set T) and picked by our algorithm (set S). This limits what an adversary can do while ordering the elements of $\tilde{E}$. Intuitively, the sets in SAMP-ALG denote the following:

- N' denotes the new elements to be added to the independent set.
- T' are the elements of T for which we still haven't read the random bit Ψ.
- S' are the elements $e \in T$ for which we have read Ψ and they turn out to be picked, i.e., $\Psi(e) = 1$.

6.6.2 Proof of the Sampling Lemma

By Lemma 6.6.1, it suffices to prove that given the preconditions of the Sampling Lemma, SAMP-ALG produces an output of expected size at least $\frac{p}{1+p}|\tilde{I}|$. More precisely, we need to show that if Ψ in SAMP-ALG is chosen as $\Psi(e) \sim \text{Bern}(1-p)$ i.i.d. for all $e \in T$, we have $\mathbb{E}_\Psi[|N'|] \geq \frac{p}{1+p}|\tilde{I}|$.

The main idea of the proof is to argue that before every iteration of the for-loop in Line 2, there are "sufficient" number of elements that are still to arrive and can be added to our solution. To achieve this, we define a set I', which intuitively denotes the set of OPT elements that are still

Algorithm SAMP-ALG

Input: $\mathcal{M}_1, \mathcal{M}_2, T$, and random bits $\Psi \in \{0,1\}^{|T|}$.

1: Initialize: N', S' to $\emptyset$, and $T' = T$
2: **for** each element $e \in \tilde{E}$ **do**
3: **if** $T \cup N' \cup e \in \mathcal{M}_2$ **then**
4: Let $C \leftarrow C_1(S' \cup N' \cup T', e) \cap T'$ ▷ *Unread elements of the formed circuit*
5: **for** each element $f \in C$ **do**
6: $T' \leftarrow T' \setminus f$
7: **if** $\Psi(f) = 1$ **then** ▷ *Auxiliary random bits* Ψ
8: $S' \leftarrow S' \cup f$ ▷ *Already picked elements*
9: **else**
10: $N' \leftarrow N' \cup e$ ▷ *Newly picked elements*
11: **Break**
12: **return** N'

to arrive and can be added to the current solution. The properties of I' are rigorously captured in Invariant 6.6.2, where $\tilde{E}_r$ denotes the remaining elements of $\tilde{E}$ that are still to be considered in the for-loop. Due to Lemma 6.6.1, this also denotes the elements of $\tilde{E}$ that are still to arrive for GREEDY. Starting with $I' = \tilde{I}$ at the beginning of SAMP-ALG, we wish to maintain the following.

Invariant 6.6.2 *For given sets S', N', T, and $\tilde{E}_r \subseteq \tilde{E}$, we have set I' satisfying this invariant if*

$$S' \cup N' \cup I' \in \mathcal{M}_1 \tag{6.8}$$

$$T \cup N' \cup I' \in \mathcal{M}_2 \tag{6.9}$$

$$I' \subseteq \tilde{E}_r \tag{6.10}$$

As the algorithm SAMP-ALG progresses, set I' has to drop some of its elements so that it continues to satisfy Invariant 6.6.2. These drops from I' are rigorously captured in Updates 6.6.3. Note that set I' and Updates 6.6.3 are just for analysis purposes, and never appear in the actual algorithm. Starting with $I' = \tilde{I}$ at the beginning of SAMP-ALG and satisfying Invariant 6.6.2, in Claim 6.6.4 we prove that Updates 6.6.3 to I' ensure that the invariant is always satisfied. This lets us use induction to prove in Claim 6.6.5 that Updates 6.6.3 never drop too many elements from I' and SAMP-ALG returns an independent set of large size.

Updates 6.6.3 *We perform the following updates to I' whenever* SAMP-ALG *reaches Line 8 or Line 10. Claim 6.6.4 shows that these updates are well-defined.*

- *Line 8: If circuit $C_1(S' \cup N' \cup I' \cup f)$ is non-empty then remove an element from I' belonging to $C_1(S' \cup N' \cup I' \cup f)$ to break the circuit.*
- *Line 10: If circuit $C_1(S' \cup N' \cup I' \cup e)$ is non-empty then remove an element from I' belonging to $C_1(S' \cup N' \cup I' \cup e)$ to break the circuit. If $C_2(T \cup N' \cup I' \cup e)$ is non-empty then remove another element from I' belonging to $C_2(T \cup N' \cup I' \cup e)$ to break the circuit. In the special case where $e \in I'$, we remove e from I'.*

The following claim (proof deferred to Section 6.6.4) shows that Updates 6.6.3 maintain Invariant 6.6.2.

Claim 6.6.4 *Given matroids $\mathcal{M}_1, \mathcal{M}_2$, a set $T \in \mathcal{M}_1 \cap \mathcal{M}_2$, a set $\tilde{E}_r \subseteq \mathsf{span}_1(T)$ (denoting the set of remaining elements), and $\Psi(e) \sim$ Bern$(1-p)$ i.i.d. for all $e \in T$, suppose there exists a set I' satisfying Invariant 6.6.2 at the beginning of some iteration of the for-loop in Line 2 of* SAMP-ALG. *Then*

(i) Updates 6.6.3 are well-defined.

(ii) Updates 6.6.3 ensure that Invariant 6.6.2 hold at the end of the iteration.

Finally, we use Invariant 6.6.2 to prove the main claim.

Claim 6.6.5 *Given matroids $\mathcal{M}_1, \mathcal{M}_2$, a set $T \in \mathcal{M}_1 \cap \mathcal{M}_2$, a set $\tilde{E}_r \subseteq \tilde{E} \subseteq \mathsf{span}_1(T)$ (denoting the set of remaining elements), and $\Psi(e) \sim$ Bern$(1-p)$ i.i.d. for all $e \in T$, suppose there exists a set I' satisfying Invariant 6.6.2 at the beginning of some iteration of the for-loop of Line 2 in* SAMP-ALG. *Then the value of N' at the end of* SAMP-ALG *satisfies*

$$\mathbb{E}_\Psi[|N'|] \geq \frac{p}{1+p}|I'|$$

Proof: To prove the claim we use induction on $|I'|$ where $I' \subseteq \tilde{E}$. WLOG we can assume that e is the first element such that C in Line 4 is non-empty. Let $C = \{t_1, \ldots, t_l\}$ where $l \geq 1$. For $j \in \{0, \ldots, l-1\}$, define event B_j as $\Psi(t_1) = \Psi(t_2) = \cdots = \Psi(t_j) = 1$ and $\Psi(t_{j+1}) = 0$. Also, define $\bar{B}$ as $\Psi(t_1) = \ldots \Psi(t_l) = 1$.

Base Case: Since C is a non-empty circuit, we can assume that any element $f \in C$ satisfies the condition $\Psi(f) = 0$ with probability p. Hence, $|N'| \geq 1$ with probability at least p, proving the required claim.

Induction Step: The events $B_0, \ldots, B_{l-1}$, and $\bar{B}$ partition the entire probability space.

Case 1 (Event B_j) : Since applying the Updates 6.6.3 preserves Invariant 6.6.2 by Claim 6.6.4, we can apply the induction hypothesis to the updated set I'. Moreover. Updates 6.6.3 remove at most $j+2$ elements from I' in the event B_j. Applying the Induction hypothesis, we can conclude that $\mathbb{E}_\Psi[|N'| \,\big|\, B_j] \geq 1 + \frac{p}{1+p}(|I'| - j - 2)$.

Case 2 (Event $\bar{B}$): Since applying the Updates 6.6.3 preserves Invariant 6.6.2 by Claim 6.6.4, we can apply the induction hypothesis to the updated set I'. Moreover, Updates 6.6.3 remove l elements from I' in the event $\bar{B}$. Conditioned on the event $\bar{B}$ and applying the induction hypothesis to the updated set I', we can conclude $\mathbb{E}_\Psi[|N'|] \geq \frac{p}{1+p}(|I'| - l)$.

Combining both the cases, we have $\mathbb{E}_\Psi[|N|]$ is at least

$$\begin{aligned}
&\sum_{j=0}^{l-1} \Pr[B_j] \cdot \mathbb{E}_\Psi[|N'| \,\big|\, B_j] + \Pr[\bar{B}] \cdot \mathbb{E}_{\bar{B}}[|N| \,\big|\, \bar{B}] \\
&\geq \sum_{j=0}^{l-1} (1-p)^j \, p \left(1 + \frac{p}{1+p}(|I'| - 2 - j)\right) + (1-p)^l \left(\frac{p}{1+p}(|I'| - l)\right) \\
&= \frac{p}{1+p}\,|I'| \qquad \text{using} \sum_{j=0}^{l-1} j\,(1-p)^j = -\frac{l\,(1-p)^l}{p} - \frac{(1-p)}{p^2}((1-p)^l - 1).
\end{aligned}$$

■

To finish the proof of Lemma 6.5.7, we start with $I' := \tilde{I}$, $T' := T$, $N' := \emptyset$, and $S' := \emptyset$ in Claim 6.6.5. The preconditions hold true because $T \cup I \in \mathcal{M}_2$, $T \in \mathcal{M}_1$, and $I \in \mathcal{M}_1$.

6.6.3 Proof of the Alternate View of Sampling Lemma

We restate the lemma for convenience.

Lemma 6.6.1. *Given a fixed Ψ and assuming the elements of $\tilde{E}$ are presented in the same order, the output of* SAMP-ALG *is the same as the output of* GREEDY$(\mathcal{M}_1/S, \mathcal{M}_2/T, \tilde{E})$.

Starting with $S' = \emptyset$, $N' = \emptyset$, and $T' = T$, we make some simple observations and prove a small claim before proving Lemma 6.6.1.

Observation 6.6.6 *The for-loop defined in Line 2 of* SAMP-ALG *maintains the following invariant*

$$S \subseteq S' \cup T' \subseteq T$$

Proof: To show the first containment, observe that for each element if an $\Psi(e) = 1$ then it simply moves from T' to S'. Hence, all the elements of $S \subseteq S' \cup T'$. To observe, the second containment, note that an element of T' either moves into S' or gets removed. Since T' was initialized to T, the second containment follows. ■

Observation 6.6.7 *The for-loop defined in Line 2 of* SAMP-ALG *maintains the following invariant*

$$S' \cup N' \cup T' \in \mathcal{M}_1.$$

Proof: Since $T \in \mathcal{M}_1$ and $S' = T' = \emptyset$ at the beginning, we can conclude that this is correct at the beginning of SAMP-ALG. Now consider an iteration of the for-loop defined in Line 2. When an element f is added to S' in Line 8, it must have belonged to T', implying that $S' \cup N' \cup T'$ is unchanged. If an element e is added to N' (in Line 10) then we must remove an element f from T' (due to Line 6), which belonged to the unique circuit $C_1(S' \cup T' \cup N', e)$. Hence, $S' \cup N' \cup e \cup (T' \setminus f)$ is still an independent set in $\mathcal{M}_1$. ■

Claim 6.6.8 *For an element $e \in \tilde{E}$, if Line 4 of* SAMP-ALG *is reached then $C_1(S' \cup N' \cup T', e)$ is non-empty.*

Proof: We know $\tilde{E} \subseteq \mathsf{span}_1(T)$. Moreover, $S' \cup T' \subseteq T \subseteq \mathsf{span}_1(T)$ (using Observation 6.6.6). Hence, $S' \cup T' \cup \tilde{E} \subseteq \mathsf{span}_1(T)$ implies

$$\mathsf{rank}_{\mathcal{M}_1}(S' \cup T' \cup \tilde{E}) \leq |T|. \tag{6.11}$$

We prove the lemma by contradiction and assume circuit $C_1(S' \cup N' \cup T', e)$ is empty. Using Observation 6.6.7, this implies $(S' \cup N' \cup T' \cup e) \in \mathcal{M}_1$. Now, $\mathsf{rank}_{\mathcal{M}_1}(S' \cup N' \cup T' \cup e) = |S' \cup N' \cup T'| + 1 \leq \mathsf{rank}_{\mathcal{M}_1}(S' \cup T' \cup \tilde{E}) \leq |T|$ using Eq. (6.11). In the next paragraph, we show that the algorithm always maintains $|S' \cup N' \cup T'| = |T|$, which gives the contradiction $|T| + 1 \leq |T|$.

To prove $|S' \cup N' \cup T'| = |T|$, we note that the only time T' decreases is in Line 6. In this case, we either add the dropped element to S' in Line 8 or a new element to N' in Line 10. Hence,

the $|S' \cup N' \cup T'|$ is unchanged in the for-loop of Line 2. Since we initialize $S' = N' = \emptyset$ and $T' = T$, we can conclude that this $|S' \cup N' \cup T'| = |T|$ is maintained. ■

We now have the tools to prove Lemma 6.6.1.

Proof:[Lemma 6.6.1] Let us assume the elements of $\tilde{E}$ are presented in order $e_1, \ldots, e_t$ where $t = |\tilde{E}|$. We will use induction on the following hypothesis.

Induction Hypothesis (I.H.): After both algorithms have seen the first k elements $e_1, \ldots, e_k$, the set N' in SAMP-ALG is the same as the set of elements selected by GREEDY$(\mathcal{M}_1/S, \mathcal{M}_2/T, \tilde{E})$.

Base Case: Initially, both algorithms have not selected any element. Hence, $N' = \emptyset$ is the set of all elements selected by GREEDY.

Induction Step: Suppose the I.H. is true for elements $e_1, \ldots, e_{k-1}$ and we are considering element e_k. If e_k does not satisfy $T \cup N' \cup e_k \in \mathcal{M}_2$, then it will also not satisfy the same condition for GREEDY because N' is the set selected by GREEDY (by I.H.) and $N' \cup e \notin \mathcal{M}_2/T$. In this case we are done with the induction step. From now assume $T \cup N' \cup e_k \in \mathcal{M}_2$.

Suppose e_k is added to N' in SAMP-ALG, then we claim GREEDY$(\mathcal{M}_1/S, \mathcal{M}_2/T, \tilde{E})$ will also select this element. The only location where e_k could be added is Line 10. This occurs when we remove some appropriate element $f \in T'$ to ensure $S' \cup (T' \setminus f) \cup N' \cup e \in \mathcal{M}_1$. Furthermore $\Psi(f) = 0$ implies $f \notin S$. By Observation 6.6.6, set $S \subseteq S' \cup T' \setminus f$. Hence, $S' \cup (T' \setminus f) \cup N' \cup e \in \mathcal{M}_1$ implies $S \cup N' \cup e \in \mathcal{M}_1$ and GREEDY will also select this element.

Next, suppose e_k is not picked by the algorithm. By Claim 6.6.8, we know that $C_1(S' \cup N' \cup T', e)$ is non-empty. In this case, every element $f \in C$ encountered in the for-loop of Line 5 must have had $\Psi(f) = 1$. This implies that at the end of the for-loop of Line 5, circuit $C_1(S' \cup N' \cup T', e) \subseteq S' \cup N'$. Since $S' \subseteq S$ (by Observation 6.6.6), this gives $N' \cup e \notin \mathcal{M}_1/S$. Hence, GREEDY cannot select element e_k. ■

6.6.4 Proof that the Updates are valid

In this section we prove Claim 6.6.4 by showing that Updates 6.6.3 are well-defined and maintain Invariant 6.6.2.

Proof:[Claim 6.6.4] Since Invariant 6.6.2 holds before entering into the for-loop in Line 2, we prove this claim by showing that after one iteration of the for-loop, i.e., after arrival of an element e, Properties (i) and (ii) hold.

We first show that the properties hold if the set C in Line 4 is empty. Since in this case we do not perform any updates to sets S', N', I', and T', Invariant 6.8, Invariant 6.9, and well-definedness trivially hold. To prove Invariant (6.10), we need to show $e \notin I'$. This is true because by Claim 6.6.8 element e forms a circuit in $C_1(S' \cup N' \cup T', e)$, and by Invariant (6.8) we know $S' \cup N' \cup I' \in \mathcal{M}_1$. Hence, the circuit $C_1(S' \cup N' \cup T', e)$ contains some element of T', which gives the contradiction that C is non-empty.

Now WLOG, we can assume that element e forms a non-empty set C in Line 4. We prove Property (i), Invariant (6.8), and Invariant (6.9) by showing that they hold after any iteration of the for-loop of Line 5. Note that sets S', N', and I' can only change in Lines 8 or 10 of the for-loop. We prove the claim for both these cases.

Case 1 (Line 8): Since f belonged to T', from Observation 6.6.7 we know $(S' \cup N' \cup f) \in \mathcal{M}_1$. Now using Invariant (6.8) (which holds before the iteration), we can deduce that $C_1(S' \cup N' \cup$

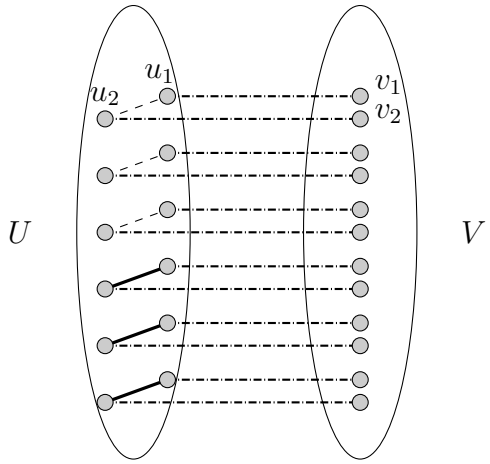

Figure 6.2: U denotes the set of vertices matched by GREEDY in Phase (a) and V denotes the remaining vertices of G. Solid edges within U denote the picked edges and dashed edges within U denote the marked ones. Dashed edges from U to V denote the OPT edges.

$I', f) \cap I'$ is non-empty and the update is well-defined. Invariant (6.8) holds because the update breaks any circuit in $S' \cup N' \cup I'$ in $\mathcal{M}_1$. Since T and N' are unchanged and I' only gets smaller, Invariant (6.9) holds trivially.

Case 2 (Line 10): Since we are adding e to N', it must be the case that $S' \cup N' \cup e \in \mathcal{M}_1$ (by Lemma 6.6.1). If $C_1(S' \cup N' \cup I' \cup e)$ is non-empty then $C_1(S' \cup N' \cup I' \cup e) \cap I'$ must be non-empty. Moreover, by Line 3, we know that $T \cup N' \cup e \in \mathcal{M}_2$. Hence, if $C_2(T \cup N' \cup I' \cup e)$ is non-empty then $C_2(T \cup N' \cup I' \cup e) \cap I'$ must be non-empty. Both of them together prove the the update is well-defined in this case. Invariant (6.8) and Invariant (6.9) hold because Updates 6.6.3 break any circuit $C_1(S' \cup N' \cup I' \cup e)$ and $C_2(T \cup N' \cup I' \cup e)$.

Finally, to finish the proof of this claim, we show that Invariant (6.10) also holds at the end of every iteration of the for-loop of Line 2. If $e \notin I'$ then Invariant (6.10) trivially holds as $\tilde{E}_r$ looses element e, and $I' \subseteq \tilde{E}_r \setminus e$. Now suppose $e \in I'$. Here we consider two cases.

Case 1 (e is added to N'): Here SAMP-ALG reaches Line 10 and the special case of Update 6.6.3 ensures that e is removed from I'. Hence $I' \subseteq \tilde{E}_r$.

Case 2 (e is not added to N'): From the above proof, we know that Invariants (6.8) and (6.9) are preserved at the end of this iteration. We prove by contradiction and assume that $e \in I'$ at the end of this iteration. Since $e \notin N'$, all the elements of C in Line 4 are added to S' by the end of this iteration. Hence, the entire circuit in Line 4 (which is non-empty by Claim 6.6.8) is contained in $S' \cup N' \cup e$ at the end of the iteration. Since $e \in I'$, this implies that $S' \cup N' \cup I'$ is not independent. This is a contradiction as Invariant (6.8) is violated. ■

6.7 Beating Half for General Graphs

Theorem 6.3.3. *In the random edge arrival model, the online matching problem for general graphs has a $(\frac{1}{2} + \delta')$-competitive randomized algorithm, where $\delta' > 0$ is a constant.*

Proof: [Proof overview] Let G be the arrival graph with edge set E. Using the same idea as Lemma 6.4.1, we can again focus on graphs where GREEDY has a competitive ratio of at most $\left(\frac{1}{2}+\epsilon\right)$ for any constant $\epsilon > 0$. We construct a two-phase algorithm that uses the algorithm from Theorem 6.1.1 as a subroutine. In Phase (a), we run GREEDY; however, each edge selected by GREEDY is picked only with probability $(1-p)$. With probability p, we mark it along with its vertices and behave as if it has been matched for the rest of Phase (a). Since the hastiness property (Lemma 6.4.3) is also true for general graphs, in expectation we pick $(1-p)\left(\frac{1}{2}-O(\frac{\epsilon}{f})\right)$ $|\text{OPT}|$ edges and mark $p\left(\frac{1}{2}-O(\frac{\epsilon}{f})\right)$ $|\text{OPT}|$ edges in Phase (a). Now we need to ensure that in expectation $(1+\gamma)$ edges, for some constant $\gamma > 0$, are picked per marked edge in Phase (b).

Let T_f denote the set of edges selected by GREEDY in Phase (a), i.e., both picked and marked edges. Let U denote the set of vertices matched in T_f and V denote the remaining set of vertices of G. Using the following simple Fact 6.7.1 and Lemma 6.4.3, we can argue that $\left(1-O(\frac{\epsilon}{f})\right)$ OPT edges go from a vertex in U to a vertex in V in graph G.

Fact 6.7.1 (Lemma 1 in [107]) *Consider a maximal matching T of graph G such that $|T| \leq \left(\frac{1}{2}+\epsilon\right)|\text{OPT}|$ for some $\epsilon \geq 0$. Then G contains at least $\left(\frac{1}{2}-3\epsilon\right)|\text{OPT}|$ vertex disjoint 3-augmenting paths with respect to T.*

Moreover, in expectation at most f fraction of these (U,V) OPT edges can appear in Phase (a). Thus, setting $\epsilon \ll f \ll 1$ gives that most of the OPT edges, i.e., $\left(1-O(\frac{\epsilon}{f})-f\right)$ fraction, appear in Phase (b). This implies that most of the marked edges contain two 3-augmentation edges as shown in Figure 6.2.

Now consider a marked edge (u_1,u_2) with (u_1,v_1) and (u_2,v_2) denoting its 3-augmentations. In comparison to bipartite graphs, the new concern in general graphs is that there might be an edge between u_1 and v_2 as triangles are possible in non-bipartite graphs. Hence, the Sampling Lemma 6.4.4 cannot be directly applied here. However, we are only interested in the bipartite graph between vertices U and V. Therefore, in Phase (b), we run the algorithm from Theorem 6.1.1 for bipartite graphs restricted to (U,V) edges. For sufficiently small values of constants ϵ and f, the constant δ gain in Theorem 6.1.1 is sufficient to obtain a constant δ' gain for this theorem. ■

6.8 Open Problems

There are several important problems in the space of online matroid problems.

Problem 6.8.1 *What is the best competitive ratio that is achievable for online matroid intersection in the random arrival model? In particular, can one achieve a ratio of $1-1/e$ for this problem?*

It is natural to consider the problem when we relax the assumption that the elements are arriving in random order. This problem is interesting even when we are dealing with the special case of online bipartite matching.

Problem 6.8.2 *Is there an algorithm with competitive ratio strictly better than half for the online biparite matching problem in the edge arrival model, when the arrivals are adversarial?*

While this work has focused on the edge arrival model, there are still some open questions in the vertex arrival model. In particular, the main open question is whether there exists a $1 - 1/e$ approximation algorithm for the *weighted* bipartite matching problem when the vertices arrive in adversarial order. We note that it is important that we allow the algorithm to throw away edges already chosen (known as the "free-disposal" assumption). Recently, Zadimoghaddam [143] showed that it is possible to break the barrier of $1/2$ barrier by a small constant. However, this is done using a complicated adaptive algorithm and a careful charging scheme. Unlike previous algorithms in this space, the charging schme increases and decreases the amount an edge is charged. It remains an open question if this can be improved further.

Chapter 7

Smoothed Online Convex Optimization

7.1 Introduction

Consider the following online optimization problem: at each time step t, the algorithm is presented with a convex function $f_t : \mathbb{R}^d \to \mathbb{R}$. The algorithm must choose a point x_t and its cost is the function cost $f(x_t)$ and the distance it must travel from the previous point $\|x_t - x_{t-1}\|$. The goal is to minimize the total cost incurred by the algorithm at all points. This problem, which is known as Smoothed Online Convex Optimization (SOCO), can be viewed as a special case of the metrical task system problem. The special case when each f_i is the indicator function for a convex set is known as the Online Convex Body Chasing Problem (OCO).

The work function algorithm (WFA) is a generic online algorithm for any metrical task system. It is optimal or is conjectured to be optimal for many classical problems including the famous k-server problem. Before, we define the work function algorithm, it is useful to consider two natural algorithms for SOCO.

Greedy Algorithm: Satisfy the request f_i using the minimal movement given current position. (This is oblivious to the past history).

Follow the Leader: Satisfy the request f_i by moving to the position where the optimal algorithm will be. (This is fully determined by the past history).

Unfortunately, it is easy to construct examples where both algorithms can perform arbitrarily badly. The work function algorithm tries to balance between these two extremes. In particular, WFA will try to move to the point z_t that minimizes the optimal cost of ending at position z_t and the distance of z_t from the current point.

In more detail, the retrospective function $r_t(x)$ denotes the cost of satisfying the first t requests (i.e. functions) and ending at position x. Formally, given a set of convex functions $f_i : \mathbb{R} \to \mathbb{R}_{\geq 0}$ for all $i \in [t]$, define the work functions as follows:

$$r_0(x) = \|x\| \tag{7.1}$$

$$r_t(x) = \min_s r_{t-1}(s) + f_{t-1}(s) + \|x - s\| \tag{7.2}$$

r_t is a convex function; this follows from the convexity of the norm $\|x\|$ and f_t. Define the work function algorithm WFA as follows. Let z_t indicate the position of the algorithm at time i (with

$z_0 = 0$). Then,

$$z_t = \text{argmin}_s \, w_{t-1}(s) + f_t(s) + \|z_{t-1} - s\|. \tag{7.3}$$

In this work, we will try and investigate the competitive ratio achieved by WFA.

7.2 Related Work

The metrical tasks system problem was introduced by Borodin et al. [28] and has received intense scrutiny. Smoothed Convex Optimization was first studied by Andrew et al. [8] and Lin et al. [117]. A randomized 2-competitive algorithm was given for this problem by Bansal et al. [20].

The problem of convex body chasing was introduced by Friedman and Linial [64] who showed that there exists a $O(1)$ algorithm for this problem when $d = 2$. The case when $d \geq 3$ remains open to this day. They also showed a lower bound of $\Omega(d^{1-1/p})$ when the underlying metric is measured with respect to the ℓ_p-norm. For the case $p = 1$, there exists a $\Omega(\log d)$ lower bound due to the work of Buchbinder et al. [29].

However, special cases of this problem have been studied recently. Antoniadis et al. [9] gave an intuitive algorithm for chasing lower dimensional objects and showed reductions between online function chasing and online convex body chasing algorithms. Recently, Bansal et al. [21] and [10] showed that convex bodies are chasable.

7.3 Our Results

The main result of this section is that

Theorem 7.3.1 *WFA is a 3-competitive for SOCO when the dimension is 1. Furthermore, this is tight.*

For higher dimensions, we show that the WFA can be viewed as minimizing the bregman divergence with respect to the retrospective function. We generalize the potential to higher dimensions and argue some natural properties of r_t. We show that showing a $O_d(1)$-competitiveness boils down to a smoothness condition on the dual of the retrospective function $r^*(x)$.

7.4 One Dimensional Case

The retrospective function $r_i(x)$ denotes the cost of satisfying the first i requests (i.e. functions) and ending at position x. r_i is a convex function; this follows from the convexity of the norm $\|x\|$ and f_i. Hence, its derivative is monotone and well defined at all but a countable number of points. For the sake of the exposition, we will assume that the derivative is well defined at all points.

Define the set $F_i = cl\big(\{x \mid |w'_{i-1}(x) + f'_i(x)| < 1\}\big)$, where cl denotes the closure operator. (The fact that F_i is a connected set follows as a $w'_{i-1}(x) + f'_i(x)$ is a monotone function).

Claim 7.4.1 *Define $\tilde{w}_i$ as follows:*

$$\tilde{w}_i(x) = \begin{cases} w_{i-1}(x) + f_i(x) & \text{if } x \in \mathcal{F}_i \\ w_{i-1}(s) + f_i(s) + |x - s| & \text{else where } s = \operatorname{argmin}_{s \in F_i} |x - s| \end{cases}$$

Then $\tilde{w}_i(x) = w_i(x)$ for all $x \in \mathbb{R}$.

It is worth understanding the above claim. In short it says that the work function only depends only on the work function value in the set F_i. Intuitively, F_i is the region where optimal solution can "move" without paying the full movement cost. In particular, for any point $y \in F_i$ and the optimal solution x_i^*, the work function grows strictly slower than the norm; I.e. $\|y - x^*\| > r(y) - r(x^*)$.

The work function algorithm exploits this fact and will ensure that each point z_t lies in the set F_i.

Claim 7.4.2 *After any set of requests i, WFA will end at a point $z_i \in F_i$.*

To understand the interplay between the sets F_i and the WFA, we will define a potential function.

Definition 7.4.3 *Define the potential*

$$\Phi_i = 2 \int_{a_i}^{b_i} (1 - |r_i'(x)|)dx + |\int_{z_i}^{o_i} w_i'(x)dx| \tag{7.4}$$

where the region $F_i = [a_i, b_i]$ and o_i denotes the minimum of r_i (i.e. the optimal solution after i requests).

To understand this better, we will split the potential $\Phi_i = \mathcal{V}_i + \mathcal{D}_i$ where the constituent parts are

- 'Volume': $\mathcal{V}_i = 2 \int_{a_i}^{b_i} (1 - |r_i'(x)|)dx$. This defines the maximum amount the optimal algorithm can move without paying its true movement cost.
- 'Distance': $\mathcal{D}_i = |\int_{z_i}^{o_i} r_i'(x)dx|$. This relates how far the algorithm is from the optimal point as measured by the work function.

In figure 7.4, we give a pictorial representation of this function.

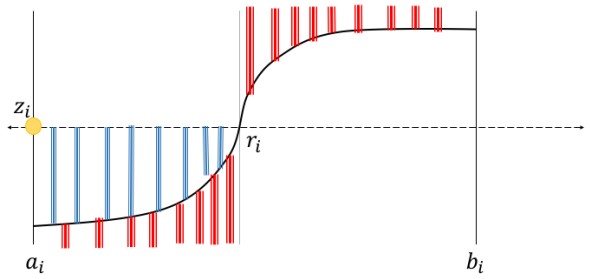

Figure 7.1: The curve represents r_i' and the dotted line is the x-axis. The red bars indicate $\frac{1}{2}\mathcal{V}_i$ and $\mathcal{D}_i$ is shown in blue.

The following lemma shows that WFA is 3-competitive.

Lemma 7.4.4 *The algorithm maintains the invariant at each step i*

$$\begin{aligned} \Phi_i &\geq 0 \qquad \forall i \geq 0 \\ 3r_i(o_i) &\geq \Phi_i + \mathit{WFA}_i \qquad \forall i \geq 0 \end{aligned}$$

where WFA_i refers to total cost incurred to serve the first i requests.

Proof:[Proof of lemma 7.4.4] We proceed by induction.

The base case: Since $o_0 = z_0 = 0$, this relationship is true when $i = 0$.

The inductive step: Suppose the invariant holds at time i. We will assume that we are given a piecewise linear function f_{i+1}. In particular, we assume[1]

$$f_{i+1}(x) = \begin{cases} (c - x) \cdot \Delta & x \leq c \\ 0 & x > c \end{cases}$$

Observe that f_{i+1} has nonpositive derivative everywhere. Since the derivative only takes on two values $-\Delta$ and 0, we can calculate precisely this can only move the region F_i to the right. In particular, $a_{i+1} \geq a_i$ and $b_{i+1} \geq b_i$.

To verify the invariant, we need to know the position c and z_i. We first consider the case when $c = b_{i+1}$. In other words, c is to the right of F_i.

First, we will analyze the change in $r_i(o_i)$. It is easy to see that

$$r_{i+1}(o_{i+1}) - r_i(o_i) = \int_{o_i}^{o_{i+1}} r_i' + (b_{i+1} - o_{i+1})\Delta.$$

This is simply the cost of satisfying the first i requests and ending at position $r_i(o_{i+1})$ and then $f_{i+1}(o_{i+1}) = (b_{i+1} - o_{i+1})\Delta$. Now we will analyze the change in $\mathcal{V}_i$.

$$\begin{aligned} \mathcal{V}_{i+1} - \mathcal{V}_i &= 2\Big(- \int_{a_i}^{a_{i+1}} (1 - |r_i'|) - \Delta(o_i - a_{i+1}) + \int_{o_i}^{o_{i+1}} (2r_i' - \Delta) + \Delta(b_{i+1} - o_{i+1})\Big) \\ &= 2\Big(- \int_{a_i}^{a_{i+1}} (1 - |r_i'|) + \Delta(b_{i+1} + a_{i+1} - 2 \cdot o_{i+1}) + \int_{o_i}^{o_{i+1}} 2r_i'\Big) \end{aligned}$$

We show that this is true in three important cases:

Case 1 $z_i = a_i$ and $c = b_{i+1}$.

$$\begin{aligned} \mathcal{V}_{i+1} - \mathcal{V}_i &= 2\Big(- \int_{a_i}^{a_{i+1}} (1 - |r_i'|) + \Delta(b_{i+1} + a_{i+1} - 2 \cdot o_{i+1}) + \int_{o_i}^{o_{i+1}} 2r_i'\Big) \\ \mathcal{D}_{i+1} - \mathcal{D}_i &= - \int_{a_i}^{a_{i+1}} |r_i'| + \Delta(o_i - a_{i+1}) + \int_{o_i}^{o_{i+1}} (\Delta - r_i') \\ cost_{i+1} - cost_i &= (a_{i+1} - a_i) + \Delta(b_{i+1} - a_{i+1}) \end{aligned}$$

[1]I think this is without loss of generality but need to double check. Either way, this is far more informative than the general case.

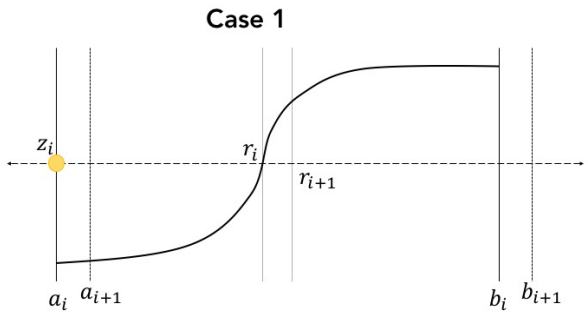

Figure 7.2: Case 1

Adding the first three equations, we get

$$\begin{gathered}
-2\int_{a_i}^{a_{i+1}}(1-|r_i'|)-\int_{a_i}^{a_{i+1}}|r_i'|+(a_{i+1}-a_i) \\
-2\Delta(o_{i+1}-a_{i+1})+\Delta(o_{i+1}-a_{i+1})+\Delta(o_{i+1}-a_{i+1}) \\
+\int_{o_i}^{o_{i+1}}4r_i'-\int_{o_i}^{o_{i+1}}r_i' \\
+2\Delta(b_{i+1}-o_{i+1})+\Delta(b_{i+1}-o_{i+1})
\end{gathered}$$

Simplifying these equations, we get

$$3\int_{o_i}^{o_{i+1}}r_i'+3(b_{i+1}-o_{i+1})\Delta)-\int_{a_i}^{a_{i+1}}(1-|r_i'|)$$

which is less than $3(r_{i+1}(o_{i+1})-r_i(o_i))$.

Case 2 $z_i = b_i$ and $c = b_{i+1}$.

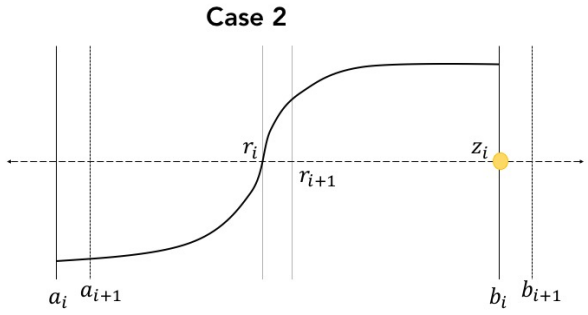

Figure 7.3: Case 1

In this case, the $\mathcal{V}_{i+1}$ and $r_{i+1}(o_{i+1})$ changes in the same way as earlier. Therefore, we can say

$$\mathcal{D}_{i+1} - \mathcal{D}_i = -\int_{o_i}^{o_{i+1}} |r_i'| - \Delta(b_i - o_{i+1})$$
$$cost_{i+1} - cost_i = \Delta(b_{i+1} - b_i)$$

Adding the first three equations, we get

$$-2\int_{a_i}^{a_{i+1}} (1 - |r_i'|)$$
$$-2\Delta(o_{i+1} - a_{i+1})$$
$$+\int_{o_i}^{o_{i+1}} 4r_i' - \int_{o_i}^{o_{i+1}} r_i'$$
$$+2\Delta(b_i - o_{i+1}) - \Delta(b_i - o_{i+1})$$
$$+2\Delta(b_{i+1} - b_i) + \Delta(b_{i+1} - b_i)$$

This value is at most $3(r_{i+1}(o_{i+1}) - r_i(o_i))$.
Case 3 $o_i \leq z_i \leq o_{i+1}$ and $c = b_{i+1}$.

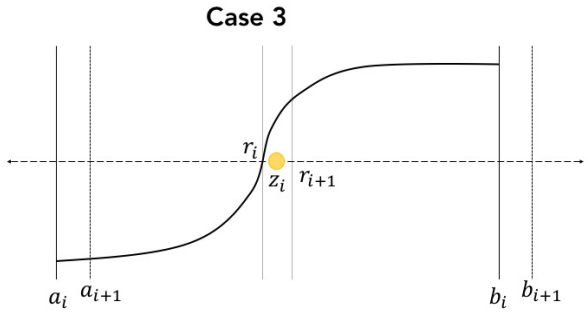

Figure 7.4: Case 3

In this case, the $\mathcal{V}_{i+1}$ and $r_{i+1}(o_{i+1})$ changes in the same way as earlier. Therefore, we can say

$$\mathcal{D}_{i+1} - \mathcal{D}_i = -\int_{o_i}^{z_i} |r_i'| + \int_{z_i}^{o_{i+1}} (\Delta - |r_i'|)$$
$$cost_{i+1} - cost_i = \Delta(b_{i+1} - z_i)$$

Adding the first three equations, we get

$$\begin{gathered}
-2\int_{a_i}^{a_{i+1}}(1-|r_i'|) \\
-2\Delta(z_i-a_{i+1}) \\
-2\Delta(o_{i+1}-z_i)+\Delta(o_{i+1}-z_i)+\Delta(o_{i+1}-z_i) \\
+\int_{o_i}^{o_{i+1}}4r_i'-\int_{o_i}^{o_{i+1}}r_i' \\
+2\Delta(b_{i+1}-o_{i+1})+\Delta(b_{i+1}-o_{i+1})
\end{gathered}$$

■

7.5 Higher Dimensions

To solve the problem of SOCO in higher diemsnions it suffices to solve the case of half-space chasing. This is a special case where each f_i is an indicator function of a halfspace. We will analyze the WFA in this case and extract some properties of the restrospecitive which will be useful in extending the potential function to higher dimensions.

7.5.1 Alternate view of Work Function Algorithms

In this section, we will show that the work function algorithm can be viewed as minimizing the bregman divergence with respect to the retrospective function. Recall that the bregman divergence with respect to a convex function f is $\mathcal{D}_f(x,y) = f(x) - f(y) - \langle \nabla f(y), x-y\rangle$.

The work function algorithm simply calculates the the minimum feasible point with respect to the retrospective.

Lemma 7.5.1 *If z_t is an optimal solution to the problem, then*

$$z_t = \operatorname{argmin}_{\substack{x \\ \langle a_i, x\rangle \geq b_i}} \mathcal{D}_{r_t}(x, z_{t-1})$$

Proof: Observe that $\|z_t - z_{t-1}\| = r_t(z_{t-1}) - r_t(z_t)$. Furthermore, we know that the optimal solution to $r_t(z_{t-1})$ uses a path that goes through $z_t \to z_{t-1}$. Therefore the derivative

$$\nabla r_t(z_{t-1}) = \frac{z_{t-1} - z_t}{\|z_{t-1} - z_t\|}$$

Puting these together, it is easy to see that $\mathcal{D}_{r_t}(z_t, z_{t-1}) = 0$. ■

Claim 7.5.2 *If x_t^* be the optimal solution $r_t(x)$ and let y be the point at which it took the previous request. I.e. the optimal path is of the form $y \to x_t^*$. Then*

$$\mathcal{D}_{r_{t-1}}(x_t^*, y) = 0$$

Proof: The first observation is that $r_t(x_t^*) = r_{t-1}(x_t^*)$. Therefore, we know that $r_{t-1}(x_t^*) - r_{t-1}(y) = \|x_t^* - y\|$. Furthermore, we know that

$$\nabla r_{t-1}(x_t^*) = \frac{x_t^* - y}{\|x_t^* - y\|}.$$

Plugging this into the definition of the bregman divergence, we get the required claim. ■

7.5.2 Fenchel Duals

The retrospective function (i.e. work function) for this problem is the solution to a convex program:

$$\begin{aligned} r_t(x) := \min \quad & \sum_{i=1}^{t} \|x_i - x_{i-1}\| + \|x - x_t\| \\ & \langle a_j, x_j \rangle \geq b_j \qquad\qquad \forall j \in [t] \end{aligned}$$

Recall the fenchel dual is defined as $r_t^*(s) := \sup_x \langle s, x \rangle - r_t(x)$

Claim 7.5.3 *$r_t^*(s)$ is equal to the solution to the following convex program*

$$\begin{aligned} \min \quad & -\sum_{i=1}^{t} \lambda_i b_i \\ \|s + \sum_{i=j}^{t} \lambda_i a_i\| \leq 1 \qquad & \forall j \in [t] \\ \|s\| \leq 1,\ \lambda \geq 0 & \end{aligned}$$

Proof: We proceed as follows:

$$\begin{aligned} r_t^*(s) &= \sup_x \langle s, x \rangle - r_t(x) \\ &= \sup_x \langle s, x \rangle - \min_{\substack{x_1,\ldots,x_t \\ \langle a_j, x_j \rangle \geq b_j}} \sum_{i=1}^{t} \|x_i - x_{i-1}\| + \|x - x_t\| \\ &= \sup_x \langle s, x \rangle - \min_{\substack{x_1,\ldots,x_t \\ \langle a_j, x_j \rangle \geq b_j}} \max_{\substack{s_1,\ldots,s_{t+1} \\ \|s_j\| \leq 1}} \sum_{i=1}^{t} \langle s_i, x_i - x_{i-1} \rangle + \langle s_{t+1}, x - x_t \rangle \end{aligned}$$

By Sion's theorem

$$\begin{aligned} &= \sup_x \langle s, x \rangle - \max_{\substack{s_1,\ldots,s_{t+1} \\ \|s_j\| \leq 1}} \min_{\substack{x_1,\ldots,x_t \\ \langle a_j, x_j \rangle \geq b_j}} \sum_{i=1}^{t} \langle s_i - s_{i+1}, x_i \rangle + \langle s_{t+1}, x \rangle \\ &= \sup_x \langle s, x \rangle - \max_{\substack{s_1,\ldots,s_{t+1} \\ \|s_j\| \leq 1}} \sum_{i=1}^{t} \min_{\substack{x_i \\ \langle a_i, x_i \rangle \geq b_i}} \langle s_i - s_{i+1}, x_i \rangle + \langle s_{t+1}, x \rangle \end{aligned}$$

Unless $s_i - s_{i+1} = \lambda_i a_i$ for some $\lambda_i \geq 0$, the minimum inside the sum will be $-\infty$.

$$= \sup_x \langle s, x \rangle - \max_{\substack{s_1,\ldots,s_{t+1} \\ s_j - s_{j+1} = \lambda_j a_j \\ \|s_j\| \leq 1, \lambda_j \geq 0}} \sum_{i=1}^{t} \lambda_i b_i + \langle s_{t+1}, x \rangle$$

$$= \min_{\substack{s_1,\ldots,s_{t+1} \\ s_j - s_{j+1} = \lambda_j a_j \\ \|s_j\| \leq 1, \lambda_j \geq 0}} \sup_x \langle s - s_{t+1}, x \rangle - \sum_{i=1}^{t} \lambda_i b_i$$

Unless $s = s_{t+1}$, the first quantity is ∞. Rewriting the above, we get the claim as desired. ■

Corollary 7.5.4 *The fenchel dual r_t^* satisfies the relationship*

$$r_t^*(s) = \min_{\substack{\lambda \geq 0 \\ \|s + \lambda a_t\| \leq 1}} r_{t-1}^*(s + \lambda a_t) - \lambda b_t$$

7.5.3 Slack

An important quantity to consider is the Slack which tells us how much the optimal solution can "cheat" by rewriting history to move to a point x from x_t^*.

Definition 7.5.5 *The Slack at a position x is $\textsf{Slack}(x) := \|x - x_t^*\| - (w_t(x) - w_t(x_t^*))$.*

Since we want to understand how this Slack over the entire metric space, we also define the directional slack, which is the supremum of the slack over points in some direction.

Definition 7.5.6 *Given a unit-vector s, the directional slack is $D_t^s := \sup_{\alpha \geq 0} \textsf{Slack}(x_t^* + \alpha s)$*

Although, thid defintion seems a little odd, it has the advantage that it can be succintly by the bregman divergence in the dual.

Lemma 7.5.7 *The directional slack is exactly*

$$\sup_{\alpha \geq 0} \textsf{Slack}(x_t^* + \alpha s) = \mathcal{D}_{r_t^*}(s, 0)$$

Proof: Rewriting this quantity, we get

$$\begin{aligned}
\sup_{\alpha \geq 0} \textsf{Slack}(x_t^* + \alpha s) &= \sup_{\alpha \geq 0} \alpha - \big(r_t(x_t^* + \alpha s) - r_t(x_t^*)\big) \\
&= \sup_{\alpha \geq 0} \alpha - \big(r_t(x_t^* + \alpha s) + r_t^*(0)\big) \\
&= \sup_{\alpha \geq 0} \alpha - \big(\max_{\tilde{s}} \langle \tilde{s}, x_t^* + \alpha s \rangle - r_t^*(\tilde{s}) + r_t^*(0)\big) \\
&= \sup_{\alpha \geq 0} \min_{\tilde{s}} r_t^*(\tilde{s}) - r_t^*(0) - \langle x_t^*, \tilde{s} \rangle + \alpha - \langle \tilde{s}, \alpha s \rangle
\end{aligned}$$

Since $\nabla r_t^*(0) = x_t^*$, we can simplify to

$$\begin{aligned}
&= \sup_{\alpha \geq 0} \min_{\tilde{s}} \mathcal{D}_{r_t^*}(\tilde{s}, 0) + \alpha(1 - \langle \tilde{s}, s \rangle) \\
&= \mathcal{D}_{r_t^*}(\tilde{s}, 0)
\end{aligned}$$

■

7.5.4 Potential Functions

Standard potential function proofs construct a non-negative function $\Phi_t \geq 0$, which satisfies $\mathsf{ALG}_t + \Phi_t \leq \alpha \cdot \mathsf{OPT}_t$ for every time step t. We will address each of these in steps:

The Potential Φ_t

Intuitively, any such function must be able to capture $\Phi_t \geq \max_x \mathsf{Slack}(x)$. This is because if the slack is large for some point, the next request could move the optimum to that location at "low" cost even though it is quite far away.

Given the characterization of Slack in terms of the dual, a reasonable potential function would be $\Phi_t = \alpha \int_{s\in\Omega} \mathcal{D}_{r_t^*}(s,0)ds$ where the integral is defined over the uniform measure on the unit-sphere. (Although, we only need a small set of properties from the unit-sphere, it is useful as a running example.)

$$\alpha \int_{s\in\Omega} \mathcal{D}_{r_t^*}(s,0)ds = \alpha\Big(\int_{s\in\Omega} \big(r_t^*(s) - r_t^*(0) - \langle x_t^*, s\rangle\big)ds\Big) = \alpha\Big(\int_{s\in\Omega} r_t^*(s)ds - r_t^*(0)\Big)$$

The Algorithm ALG_t

Instead of simply tracking ALG_t, it is more useful to track the quantity

$$\mathsf{ALG}_t + r_t(z_t).$$

By the definition of the WFA, we can say that $\mathsf{ALG}_t - \mathsf{ALG}_{t-1} = r_t(z_{t-1}) - r_t(z_t)$, which implies

$$\Delta(\mathsf{ALG}_t + r_t(z_t)) = r_t(z_{t-1}) - r_{t-1}(z_{t-1}) \leq \sup_x r_t(x) - r_{t-1}(x).$$

The latter quantity is called *Extended Cost*. In the case of half-space chasing, we can bound the extended cost by the increase in dual.

Claim 7.5.8 *We can bound* $\sup_x r_t(x) - r_{t-1}(x) \leq \sup_s r_t^*(s) - r_{t-1}^*(x)$.

Proof: Rewriting $r_t(x) = \sup_s \langle s, x\rangle - r_t^*(s)$, we can say that

$$\begin{aligned}
\sup_x r_t(x) - r_{t-1}(x) &= \sup_x \sup_s \big(\langle s, x\rangle - r^*(s)\big) - \sup_{\hat{s}} \big(\langle \hat{s}, x\rangle - r_{t-1}^*(\hat{s})\big) \\
&\leq \sup_x \sup_s \big(\langle s, x\rangle - r^*(s) - \langle s, x\rangle + r_{t-1}^*(s)\big) \\
&\leq \sup_s r_{t-1}^*(s) - r_t(s)
\end{aligned}$$

■

Combining these together

To show an α approximation it suffices to prove

$$\Delta(\Phi + \mathsf{ALG}) \leq \alpha \Delta \mathsf{OPT}.$$

Expanding the above terms and simplifying we get

$$\alpha \int_{s \in \Omega} r_t^*(s) - r_{t-1}^*(s) + \max_s r_{t-1}^*(s) - r_t^*(s) \leq 0.$$

Appendix A

Independent Sets Appendix

A.1 Johansson's Algorithm for Coloring Sparse Graphs

For completeness, we state two results of Johansson [96] on coloring degree-d graphs: one about graphs where vertex neighborhoods can be colored using few colors ("locally-colorable" graphs), and another about K_r-free graphs. Since the original manuscript is not available online, a complete proof is presented in the arXiv version of this paper [19].

Theorem A.1.1 *For any r, Δ, there exists a randomized algorithm that, given a graph G with maximum degree Δ such that the neighborhood of each vertex is r-colorable, outputs a proper coloring of $V(G)$ using $O(\frac{\Delta}{\ln \Delta} \ln r)$ colors in expected $\mathrm{poly}(n2^{\Delta})$ time.*

Theorem A.1.2 *For any r, Δ, there exists a randomized algorithm that, given a graph G with maximum degree Δ which excludes K_r as a subgraph, outputs a proper coloring of $V(G)$ using $O\big(\frac{\Delta}{\ln \Delta}(r^2 + r \ln \ln \Delta)\big)$ colors in expected $\mathrm{poly}(n)$ time.*

A.2 Miscellaneous Proofs

A.2.1 Proof of Theorem 2.4.1

Recall the statement of Halperin's theorem: for $\eta \in [0, \frac{1}{2}]$, suppose Z is the collection of vectors v_i satisfying $\|v_i\|^2 \geq \eta$ in the SDP solution. Then we want to find an independent set of size $\Omega\left(\frac{d^{2\eta}}{d\sqrt{\ln d}}|Z|\right)$.

Let $a_i = v_i \cdot v_0 = \|v_i\|^2$, and let $w_i = v_i - \langle v_i, v_0\rangle v_0$ denote the projection of v_i to $v_0^{\perp}$, the hyperplane orthogonal to v_0. As $\|w_i\|^2 + \langle v_i, v_0\rangle^2 = \|v_i\|^2$, we obtain $\|w_i\|^2 = a_i - a_i^2$. Let $u_i = w_i/|w_i|$.

Now for any pair of vertices i, j, we have that $w_i \cdot w_j = v_i \cdot v_j - \langle v_i, v_0\rangle\langle v_j, v_0\rangle$. As $v_i \cdot v_j = 0$ if $(i, j) \in E$, we have that

$$w_i \cdot w_j = v_i \cdot v_j - \langle v_i, v_0\rangle\langle v_j, v_0\rangle = -a_i a_j$$

and hence

$$u_i \cdot u_j = -\frac{\sqrt{a_i a_j}}{\sqrt{(1-a_i)(1-a_j)}} \leq -\frac{\eta}{(1-\eta)}.$$

The last step follows as $a_i, a_j \geq \eta$ and as $x/1-x$ is increasing for $x \in [0,1]$.

Thus the unit vectors u_i can be viewed as a feasible solution to a vector k-coloring (in the sense of [100]) where k is such that $1/(k-1) = \eta/(1-\eta)$. This gives $k = 1/\eta$, and now we can use the result of [100, Lemma 7.1] that such graphs have independent sets of size $\Omega(|Z|/d^{1-2/k}\sqrt{\ln d}) = \Omega(|Z|d^{2\eta}/(d\sqrt{\ln d}))$.

A.2.2 Proof of Theorem 2.6.4

The proof of Theorem 2.6.4 is similar to that of Theorem 2.5.1, and we give only the differences. We set $\gamma = \frac{\log d}{\log k}$. As in that proof, we wish to show that $\mathbb{E}[X_v] \geq c\gamma$ for each vertex v and some constant $c > 0$. The next few steps of the proof are identical, culminating in (2.6), which shows that

$$\mathbb{E}[X_v \mid W \cap V(H') = S] \geq d\frac{1}{2^{\epsilon x}+1} + \frac{\epsilon x}{10\log(1/\epsilon+1)}\frac{2^{\epsilon x}}{2^{\epsilon x}+1}.$$

Again, if $2^{\epsilon x}+1 \leq \sqrt{d}$, then the first term is at least $\sqrt{d}$ and we are done. Otherwise, it must be that $\epsilon x \geq (1/2)\log d$ and hence the right hand side in (2.6) is at least

$$\frac{\log d}{20\log(1/\epsilon+1)}. \tag{A.1}$$

We now consider two cases depending on the value of x. Recall the assumptions on the graph H: namely, for any vertex v and any subset T lying in the neighborhood of v of size at most $k\log^2 d$, there is an independent set of size at least $|T|/k$.

- If $x \leq k\log^2 d$, then by our assumption, X contains an independent set of size at least $|X|/k$. Every subset of this is also an independent set, and hence the number of independent sets in X is $2^{\epsilon x} \geq 2^{x/k}$. Hence $\epsilon \geq 1/k$, and so gives that (A.1) is at least $\frac{\log d}{40\log(k+1)}$.
- If $x \geq k\log^2 d$, then again by our assumption, X contains at least $2^{\log^2 d}$ independent sets, and hence $\epsilon x \geq \log^2 d$. As $x \leq d$, it follows that $\epsilon \geq \log^2 d/d \geq 1/d$ and hence $\log(1/\epsilon+1) \leq 2\log d$. Thus the right hand side of (2.6) is at least

$$\frac{\epsilon x}{20\log(1/\epsilon+1)} \geq \frac{\log^2 d}{40\log d}.$$

In all cases we have an independent set of size $\Omega\left(\frac{n}{d}\frac{\log d}{\log(k+1)}\right)$, which completes the proof of Theorem 2.6.4.

A.3 The Average-degree Case

In this section, we show that any algorithm for graphs with *maximum* degree d based on (lifts of) the standard SDP can be translated into an algorithm for graphs with *average* degree δ, albeit with a slight loss in performance. E.g., an integrality gap of $O(d/\log^2 d)$ translates to one of $O(\delta/\log^{1.5}\delta)$. Moreover, we show that it is unlikely that we can do better.

Lemma A.3.1 *Let $\epsilon \leq 1$ and $\ell \geq 1$. Suppose the integrality gap of the ℓ-level SA^+ semidefinite relaxation on graphs with* maximum *degree d is*

$$\widetilde{O}\left(\frac{d}{(\log d)^{1+\epsilon}}\right),$$

then its integrality gap on graphs with average *degree δ is at most*

$$\widetilde{O}\left(\frac{\delta}{(\log \delta)^{1+\epsilon/2}}\right).$$

Proof: Let sdp denote the value of the ℓ-level SA^+ semidefinite relaxation on the graph G, and define $\beta := 1/(\log^{1+\epsilon/2}\delta)$. We can assume that $\mathsf{sdp} \geq 3\beta n$: indeed, greedy algorithm finds an independent set of size at least $n/(\delta+1) \geq \mathsf{sdp}/(3\beta(\delta+1))$, and thus bounds the integrality gap by $\tilde{O}(\delta/\log^{1+\epsilon/2}\delta)$.

Define $\eta = \frac{c\log\log\delta}{\log\delta}$ and partition the vertices into three sets as follows:

$$\begin{aligned} A &= \{v \mid x_v \geq \eta\} \\ B &= \{v \mid \beta \leq x_v < \eta\} \\ C &= \{v \mid x_v < \beta\} \end{aligned}$$

Let $x(S) = \sum_{v\in S} x_v$. Since $\mathsf{sdp} \geq 3\beta n$, the SDP value of vertices in C is $x(C) < |C|\beta \leq \beta n \leq \mathsf{sdp}/3$. Hence, at least one of $x(A)$ or $x(B)$ is greater than $\mathsf{sdp}/3$. In each case, we will exhibit an independent set of size $\Omega((\log^{1+\epsilon/2}\delta)/\delta)\cdot\mathsf{sdp}$, which in turn will bound the integrality gap.

Case I: suppose $x(A) \geq \mathsf{sdp}/3$. This implies that $|A| \geq x(A) \geq \mathsf{sdp}/3 \geq \beta n$. We define a vertex v to be "A-high" if $deg(v) \geq a := \frac{2}{\beta}\delta$. By Markov's inequality, there are at most $\beta n/2$ A-high vertices in G. If we drop A-high vertices from A, there are at least $|A| - \beta n/2 \geq |A|/2$ vertices in the remaining set A'. Furthermore, the graph $G[A']$ has maximum degree a. Applying Theorem 2.4.1 to the set of vectors in the solution induced on vertices from A' gives an independent set of size

$$\Omega\left(|A'|\cdot\frac{a^{2\eta}}{a\sqrt{\log a}}\right) \geq \Omega\left(|A|\cdot\frac{(\log\delta)^{2c}}{\delta(\log\delta)^2}\right) \geq \Omega\left(\frac{(\log\delta)^{2c}}{\delta(\log\delta)^2}\right)\mathsf{sdp}$$

Setting $c \geq 3/2 + \epsilon/4$ completes this case.

Case II: suppose $x(B) \geq \mathsf{sdp}/3$. This implies that $|B|\cdot\eta \geq x(B) \geq \mathsf{sdp}/3 \geq \beta n$. Hence, we can say that $|B| \geq \frac{\beta}{\eta}n$. We define a vertex v to be "B-high" if $deg(v) \geq b := 2\frac{\eta}{\beta}\delta$. By Markov's inequality, there are at most $\frac{\beta}{\eta}n/2$ B-high vertices in G. If we drop B-high vertices from B, there

are at least $|B|/2$ vertices in the remaining set B'. The graph $G[B']$ now has maximum degree b. Moreover, $x(B') \geq x(B) - (\frac{\beta}{\eta} n/2) \cdot \eta \geq x(B)/2$, so the optimal value of the SA^+ relaxation on the graph $G[B']$ is at least as high. Now we can apply the assumption on the integrality gap of the convex program to $G[B']$ to infer the existence of an independent set in $G[B']$ (and hence in G) of size

$$\widetilde{\Omega}\left(\frac{b}{\log^{1+\epsilon} b} x(B')\right) = \widetilde{\Omega}\left(\frac{\delta}{\log^{1+\epsilon/2} \delta} x(B)\right) \geq \widetilde{\Omega}\left(\frac{\delta}{\log^{1+\epsilon/2} \delta} \mathsf{sdp}\right)$$

■

To show this transformation cannot be improved substantially, consider a graph G showing the integrality gap of the SDP in terms of the maximum degree d is $\Omega(d/(\log d)^{1+\epsilon})$. Specifically, let G be a graph on n vertices and maximum degree d such that $\alpha(G) = O(n\beta/d)$, but the value of the semidefinite program is $\mathsf{sdp}(G) = \Omega(n\beta/(\log d)^{1+\epsilon})$ for some $\beta \in [1, (\log d)^{\epsilon}]$ [1]. From this we construct an instance H with an integrality gap of $\widetilde{\Omega}(\delta/(\log \delta)^{1+\epsilon/2})$ where δ is the average degree.

Define $n' = n\,(\log d)^{\epsilon/2}$ and $\delta' = d/(\log d)^{\epsilon/2}$, so that $nd = n'\delta'$. We construct H by taking the union of G with n'/δ' disjoint copies of $K_{\delta'}$, the complete graph on δ' vertices. The number of edges in H is at most $nd + n'\delta' = 2nd$ and the number of vertices is $n' + n \in [n', 2n']$, the average degree of H is $\delta := O(\delta')$. Furthermore, $\mathsf{sdp}(H) \geq \mathsf{sdp}(G) = \Omega(n\beta/(\log d)^{1+\epsilon}) = \Omega(n'\beta/(\log \delta)^{1+\epsilon/2})$ and $\alpha(H) = n\beta/d + n'/\delta' \leq 2n'/\delta' = O(n'/\delta)$. Therefore, the integrality gap of the SDP on the instance H is at least $\widetilde{\Omega}(\delta/(\log \delta)^{1+\epsilon/2})$, where δ is its average degree.

[1] We may assume $\beta \geq 1$ as $\alpha(G) \geq n/d$ for any d-regular graph. If $\mathsf{sdp}(G) \geq \Omega(n/\log d)$, then Theorem 2.4.1 gives us a better integrality gap. Therefore, we can assume $\beta \in [1, (\log d)^{\epsilon}]$.

Appendix B

Online Matroid Intersection Appendix

B.1 Notation

B.2 Miscellaneous Results

B.2.1 GREEDY Beats Half on Almost Regular Graphs

Theorem B.2.1 *For online matching in random edge arrival model,* GREEDY *has a competitive ratio of at least* $(1 - \frac{1}{e})$ *on any* d*-regular graph.*

Proof: Consider a vertex v, and let $u_1, u_2, \ldots, u_d$ be its neighbours. The probability that (u_1, v) is the first to occur amongst all the edges of u_1 is exactly $\frac{1}{d}$. If this occurs, then we know that vertex v will be surely matched. Thus, the probability that v is not matched by the end of the algorithm is at most $(1 - \frac{1}{d})^d \leq \frac{1}{e}$. This means that each vertex is matched with probability at least $1 - \frac{1}{e}$, leading to the stated theorem. ■

The same analysis also extends to graphs that are almost regular, i.e., graphs with vertex degrees between $d\,(1 \pm \epsilon)$, for any small constant ϵ.

B.2.2 GREEDY Cannot Always Beat Half for Bipartite Graphs

Dyer and Frieze [49] showed a general graph[1] for which GREEDY is half competitive. Inspired from their construction, we give the following bipartite graph for which GREEDY is half competitive.

Definition B.2.2 (Thick-$\mathcal{Z}$ graph) *Let graph Thick-*$\mathcal{Z} := ((U_1 \cup U_2) \cup (V_1 \cup V_2)\}, E)$ *be a bipartite graph with* $|U_1| = |V_1|$ *and* $|U_2| = |V_2|$*. The edge set* E *consists of the union of a perfect matching between* U_i *and* V_i *for* $i \in \{1, 2\}$ *and a complete bipartite graph between* U_2 *and* V_1*. If additionally* $|U_1| = |V_2|$*, we call the graph a balanced Thick-*$\mathcal{Z}$*.*

[1]This graph is popularly known as a *bomb graph*. It is obtained by adding a new vertex and edge adjacent to each vertex of a complete graph.

General Notation

$\mathcal{M}_i$	Matroid indexed by i
$A \in \mathcal{M}$	Subset A is an independent set in the matroid $\mathcal{M}$
$T \cup e$	Short form for notation $T \cup \{e\}$
$\mathsf{rank}_{\mathcal{M}}$	The rank function defined by matroid $\mathcal{M}$
$\bar{i}$	Denotes the index $3 - i$
$\mathcal{M}_1 \cap \mathcal{M}_2$	The set of subsets that are independent in both matroids $\mathcal{M}_1$ and $\mathcal{M}_2$
$\mathcal{M}/T$	The matroid resulting from contracting subset T in matroid $\mathcal{M}$
$\mathsf{span}_i(T)$	$\{e \mid (e \in E) \wedge (\mathsf{rank}_{\mathcal{M}_i}(T \cup \{e\}) = \mathsf{rank}_{\mathcal{M}_i}(T)\}$
$C_i(T \cup e)$	The unique circuit formed by $T \cup \{e\}$ in matroid $\mathcal{M}_i$. This is undefined when T is not an independent set and $e \notin \mathsf{span}_i(T)$.
E	The set of ground elements common to the matroids $\mathcal{M}_1$ and $\mathcal{M}_2$
π	A permutation on the set E
OPT	A fixed maximum independent set in the intersection of $\mathcal{M}_1 \cap \mathcal{M}_2$
$\mathcal{G}(f)$	$\mathbb{E}_\pi[\lvert T_f \rvert]/\lvert \mathsf{OPT} \rvert$

Notation used by MARKING-GREEDY in Section 6.5.3

Ψ	The set of random bits used in the algorithm. For each $e \in E$, we have $\Psi(e) \sim Bern(1-p)$
selecting	The element is chosen by GREEDY in Phase (a)
picking	The element is chosen by MARKING-GREEDY in the final solution
marking	The element is chosen by GREEDY in Phase (a) but the algorithm does not pick it
T_f	The set of elements selected by GREEDY in Phase (a)
S	The set of elements picked by MARKING-GREEDY in Phase (a)
N_i	The set of elements belonging to $\mathcal{M}_i/S \cap \mathcal{M}_{\bar{i}}/T$ picked by MARKING-GREEDY in Phase (b)

Table B.1: Table of Notation

Lemma B.2.3 *When the edges of a balanced Thick-$\mathcal{Z}$ are revealed one-by-one in a random order to* GREEDY *then in expectation it produces a matching of size* $\left(\frac{1}{2} + o(1)\right)|\mathsf{OPT}|$.

Proof: We note that after an edge is picked by GREEDY, both the end points of the edge do not participate later in the algorithm. Hence, at any instance during the execution of GREEDY, the participating graph is still a Thick-$\mathcal{Z}$ graph $((U_1' \cup U_2') \cup (V_1' \cup V_2')\}, E')$, where $U_i' \subseteq U_i$ and $V_i' \subseteq V_i$ for $i \in \{1, 2\}$.

We can view the choices made by GREEDY as being done in time steps, where GREEDY chooses one edge at each time step. At each time step, at least one of U_1 or U_2 decrease by 1, and GREEDY halts when $|U_1'| = |U_2'| = 0$. Let t be the random variable indicating the first time step during the execution of GREEDY when $\min\{|U_1'|, |U_2'|\} = n^{2/3}$. Let a, b be the random variables denoting $a := |U_1'| = |V_1'|$ and $b := |U_2'| = |V_2'|$ at time t. Let O_1 denote the number of edges of OPT chosen by GREEDY before time t and let O_2 denote the number of edges of OPT chosen after time t.

We observe that the matching produced by GREEDY is of size $\frac{n}{2} + |O_1| + |O_2|$. Observe $|\big(|U_1'| - |U_2'|\big)|$ changes only when GREEDY chooses an edge from OPT, implying that we can bound $|a - b| \leq |O_1|$. Since O_2 is bounded by $|U_1'| + |U_2'|$ at time t, we can say

$$|O_2| \leq a + b = 2\,\min\{a, b\} + |a - b| \leq 2\,n^{2/3} + |O_1|.$$

Next, to bound $|O_1|$, we note that before time t the probability of an edge picked by GREEDY being from OPT is at most $\frac{2n}{n^{2/3} \cdot n^{2/3}} = \frac{2}{n^{1/3}}$. Since GREEDY picks at most n edges before time t, we have $\mathbb{E}[|O_1|] \leq \frac{2n}{n^{1/3}} = 2\,n^{2/3}$. This proves that expected size of the matching chosen by GREEDY is $\frac{n}{2} + \mathbb{E}[|O_1| + |O_2|] \leq \frac{n}{2} + 2\,n^{2/3} + 2\,\mathbb{E}[|O_1|] \leq \frac{n}{2} + 6\,n^{2/3}$.

■

B.2.3 Limitations on any OBME Algorithm

Lemma B.2.4 *No randomized algorithm can achieve a competitive ratio greater than* $\frac{5}{6} \sim 0.833$ *for online bipartite matching in random edge arrival model when the graph is a balanced Thick-$\mathcal{Z}$ with $n = 1$. This is true, even the adversary knows the graph and can identify one vertex which has degree* 2.

Proof: The optimum offline matching size is two. However, no randomized online algorithm, (even one which knows the input graph), can obtain more than $\frac{5}{3}$ edges in expectation over the random edge order. To see this, let p denote the probability that the algorithm picks the first edge it sees.

Case 1: The first edge is from the optimal matching (i.e. the first edge is of the form (u_i, v_i) for $i \in \{1, 2\}$). In this case, the algorithm will achieve the optimal value 2 with probability p. If it skips one of these edges, it will retain at most 1 edge in the remaining graph.

Case 2: The first edge is not from the optimal matching (i.e. the first edge is (u_1, v_2)). In this case the algorithm will achieve a value of at most $1 \cdot p + 2 \cdot (1 - p)$.

Since Case 1 occurs with probability $\frac{2}{3}$ and Case 2 occurs with probability $\frac{1}{3}$, the expected value of the algorithm is $\frac{5}{3}p + \frac{4}{3}(1 - p) \leq \frac{5}{3}$. ■

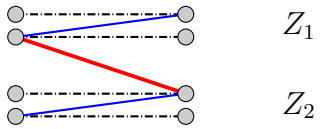

Figure B.1: The above example is a conjunction of two Thick-$\mathcal{Z}$ graphs (Z_1 and Z_2) by a single edge (the thick red edge). Note that for a Thick-$\mathcal{Z}$ graph even knowing the degree 2 vertex does not allow any algorithm to achieve more than $\frac{5}{3}$ edges in expectation.

Lemma B.2.5 *No randomized algorithm can achieve a competitive ratio greater than* $\frac{69}{84} \sim 0.821$ *for online bipartite matching in random edge arrival model.*

Proof: Our instance corresponds to the case where we take two copies of balanced Thick-$\mathcal{Z}$ graph joined by a single edge (see Figure B.1). The input is some permutation of the graphs (where the vertices or edges may be permuted and U and V may be swapped). We show by case analysis that no algorithm can achieve a competitive ratio better than $\frac{69}{84} < \frac{5}{6}$. Intuitively, the addition of the single edge only hurts any algorithm without compromising the independence between the two instances.

Let p be the probability that the algorithm picks the first edge. Consider the following cases based on Figure B.1:

Case 1: Suppose the first edge is the thick red edge. This occurs with probability $\frac{1}{7}$. If the algorithm picks this edge (which happens with probability p), then the optimal value in the remaining graph is 3. Otherwise, it can get at most $2 \cdot \frac{5}{3}$ as the two Thick-$\mathcal{Z}$ graphs are disjoint and we can use the previous lemma. Hence the expected outcome is $\frac{1}{7}\big(p \cdot 3 + (1-p) \cdot \frac{10}{3}\big)$.

Case 2: Suppose the first edge is a blue edge, this occurs with probability $\frac{2}{7}$. If the algorithm chooses this edge, then we can get value of 1. Since this affords no information about the second Z, the best an algorithm can do is $\frac{5}{3}$. Hence the expected solution is $\frac{2}{7}\big(p(1+\frac{5}{3}) + (1-p)(2+\frac{5}{3})\big)$.

Case 3: Suppose the first edge is a black edge. This occurs with probability $\frac{4}{7}$. If the algorithm chooses the first edge, then we can get value of 2 in this copy of the Thick-$\mathcal{Z}$. However, still the algorithms gets at most $\frac{5}{3}$ in the remaining copy of Thick-$\mathcal{Z}$. Hence the expected cost of the solution is $\frac{4}{7}\big(p(2+\frac{5}{3}) + (1-p)(1+\frac{5}{3})\big)$

Adding these cases together, we get that expected solution has value at most $\frac{64+5p}{21}$. Since the optimal solution is 4, this gives an upper bound of $\frac{69}{84}$.

■

B.2.4 When Size of the Ground Set is Unknown

Theorem B.2.6 *For any constant* $\epsilon > 0$*, any randomized algorithm* $\mathcal{A}$ *that does not know the number of edges to arrive has a competitive ratio* $\alpha \leq \frac{2}{3} + \epsilon$ *for online bipartite matching in random edge arrival model.*

Proof: To prove this theorem, we show that for any $\epsilon > 0$ there exists an instance where $\mathcal{A}$ is less than $\frac{2}{3} + \epsilon$-competitive.

Since $\mathcal{A}$ does not know the number of edges to arrive, it must maintain an α approximation in expectation after the arrival of every edge. This is because $\mathcal{A}$ does not know if the current edge will be the last edge.

Consider the instance given by the graph balanced Thick-$\mathcal{Z}$ (see Definition B.2.2) where the size of the $|U_1| = |V_1| = N$ will be set later. Consider a random permutation π on the set of all edges and note that each edge $e$appears in the first T edges with probability $\frac{T}{N^2+2N}$, where $T = 4(N+2)\log N$. The previous probability is at least $\frac{4\log N}{N}$. Let G_T denote the set of edges from the perfect matching between U_i and V_i that appear in the first T edges. Let B_T denote the set of edges from U_2 to V_1 that appear in the first T edges. By linearity of expectation, we can say $\mathbb{E}[|G_T|] \leq 8\log N$ and $\mathbb{E}[|B_T|] \leq 4N\log N$.

Let OPT_T denote the expected size of the maximum matching on the graph induced by the first T edges.

Claim B.2.7 $N(1-\epsilon) \leq \mathbb{E}_\pi[|\mathsf{OPT}_T|]$

Proof: Consider the graph induced between U_2 and V_1 in the first T edges. Since any particular edge occurs with probability $\frac{4\log N}{N}$ and the edges are negatively correlated, we can conclude that

$$\Pr[\exists \text{a perfect matching between } U_2 \text{ and } V_1 \text{ in the first } T \text{ edges}] \geq \Pr[\exists \text{ a perfect matching in } \mathcal{G}_{N,N,\frac{4\log N}{N}}].$$

By a result of Erdos and Renyi (see [55]), we know that

$$\lim_{N\to\infty} \Pr[\exists \text{a perfect matching in } \mathcal{G}_{N,N,\frac{4\log N}{N}}] = 1$$

Hence, we can choose an N such that the above probability is at least $1-\epsilon$. Thus we can conlude that $\mathbb{E}[\mathsf{OPT}_T] \geq N(1-\epsilon)$. ■

Let M_{OPT} denote the expected number of edges picked by $\mathcal{A}$ that belong to the perfect matching between U_i and V_i (for $i = 1, 2$) at time T. Similarly, let M_{Rest} denote the expected number of edges between U_2 and V_1 chosen by $\mathcal{A}$.

Since $\mathcal{A}$ must maintain an α approximation, we can say $M_{\mathsf{OPT}} + M_{Rest} \geq \alpha(1-\epsilon)N$. Since $M_{\mathsf{OPT}} \leq \mathbb{E}[|G_T|] = 8\log N \leq \alpha\epsilon N$, we can say

$$M_{Rest} \geq (\alpha - 2\epsilon)N \tag{B.1}$$

However, every edge chosen from M_{Rest} decreases the value of the optimal algorithm by one. Let F be the expected size of the matching chosen by the algorithm. We know that $\alpha \cdot 2N \leq F \leq 2N - M_{Rest}$. Substituting into Eq. (B.1) and dividing by $2\,N$, we get $\alpha \leq \frac{2}{3} + \epsilon$. ■

B.3 Facts

Fact 6.4.7.

$$|T_1^\pi| \geq \frac{1}{2}\left(|\mathsf{OPT}| + \sum_{e\in\mathsf{OPT}} \mathbf{1}[\text{Both ends of } e \text{ matched in } T_f^\pi]\right) \text{ and}$$

$$|T_1^\pi| \geq |T_f^\pi| + \frac{1}{2}\sum_{e\in\mathsf{OPT}} \mathbf{1}[\text{Both ends of } e \text{ unmatched in } T_f^\pi].$$

Proof: We start by counting the vertices matched in T_1^π,

$$2\,|T_1^\pi| \geq 2 \sum_{e \in \mathsf{OPT}} \mathbf{1}[\text{Both ends of } e \text{ matched in } T_1^\pi] + \sum_{e \in \mathsf{OPT}} \mathbf{1}[\text{Exactly one end of } e \text{ matched in } T_1^\pi]$$

Since T_1^π is a maximal set,

$$|OPT| = \sum_{e \in \mathsf{OPT}} \mathbf{1}[\text{Exactly one end of } e \text{ matched in } T_1^\pi] + \sum_{e \in \mathsf{OPT}} \mathbf{1}[\text{Both ends of } e \text{ matched in } T_1^\pi]$$

Combining the previous two statements and the fact that $T_f^\pi \subseteq T_1^\pi$,

$$|T_1^\pi| \geq \frac{1}{2}\left(|\mathsf{OPT}| + \sum_{e \in \mathsf{OPT}} \mathbf{1}[\text{Both ends of } e \text{ matched in } T_f^\pi]\right).$$

To prove the second part, observe that $T_f^\pi \subseteq T_1^\pi$ and T_1^π is a maximal matching. For each edge of OPT that has both its end points unmatched in T_f^π, at least one end point is adjacent to an edge T_1^π. Since these edges must be part of $T_1^\pi \setminus T_f^\pi$,

$$|T_1^\pi| \geq |T_f^\pi| + \frac{1}{2} \sum_{e \in \mathsf{OPT}} \mathbf{1}[\text{Both ends of } e \text{ unmatched in } T_f^\pi].$$

■

Fact 6.5.3. *Consider any matroid $\mathcal{M}$ and independent sets $A, B, C \in \mathcal{M}$ such that $A \subseteq \mathsf{span}_{\mathcal{M}}(B)$ and $B \cup C \in \mathcal{M}$. Then we also have $A \cup C \in \mathcal{M}$.*

Proof: Suppose we start with $B \in M$ and add elements of $A = \{a_1, a_2, \ldots, a_k\}$ one by the one. We show that one can ensure that the set remains independent in $\mathcal{M}$ by removing some elements from B. First, note that $|B| = \mathsf{rank}(B) = \mathsf{rank}(B \cup A)$. Our algorithm removes an element from B only if addition of a_j creates a circuit. Hence the rank of the set is always $|B|$ and addition of every a_j creates a unique circuit. Moreover, this circuit contains an element $b_j \in B$ that can be removed as we know $A \in \mathcal{M}$.

Next we repeat the above procedure but by starting with $B \cup C \in \mathcal{M}$ and adding elements of A. We know from before that addition of each element a_j creates a unique circuit that does not contain an element of C. Hence we can remove element b_j while ensuring the set remains independent in $\mathcal{M}$. This will finally give $A \cup C \in \mathcal{M}$. ■

B.4 Hastiness Lemma

The proof of the following lemma is similar to Lemma 2 in [107].

Lemma 6.5.1 (Hastiness Lemma). *For any two matroids $\mathcal{M}_1$ and $\mathcal{M}_2$ on the same ground set E, let T_f^π denote the set selected by* GREEDY *after running for the first f fraction of elements*

E appearing in order π. Also, for $i \in \{1,2\}$, let $\Phi_i(T_f^\pi) := \mathsf{span}_i(T_f^\pi) \cap \mathsf{OPT}$. Now for any $0 < f, \epsilon \leq \frac{1}{2}$, if $\mathbb{E}_\pi[|T_1^\pi|] \leq (\frac{1}{2} + \epsilon)\,|\mathsf{OPT}|$ then

$$\mathbb{E}_\pi\left[|\Phi_1(T_f^\pi) \cap \Phi_2(T_f^\pi)|\right] \leq 2\epsilon\,|\mathsf{OPT}| \qquad \textit{and} \tag{B.2}$$

$$\mathbb{E}_\pi\left[|\Phi_1(T_f^\pi) \cup \Phi_2(T_f^\pi)|\right] \geq \left(1 - \frac{2\epsilon}{f} + 2\epsilon\right)\,|\mathsf{OPT}|. \tag{B.3}$$

This implies $\mathcal{G}(f) := \frac{\mathbb{E}_\pi[|T_f^\pi|]}{|\mathsf{OPT}|} \geq \left(\frac{1}{2} - \left(\frac{1}{f} - 2\right)\epsilon\right)$.

Proof: For ease of notation, we write T_f^π by T_f. To prove Eq. (B.2),

$$\begin{aligned}
\mathbb{E}_\pi\left[|\Phi_1(T_f) \cap \Phi_2(T_f)|\right] &\leq \mathbb{E}_\pi\left[|\Phi_1(T_1) \cap \Phi_2(T_1)|\right] && \text{(because } T_f \subseteq T_1\text{)}\\
&= \mathbb{E}_\pi\left[(|\Phi_1(T_1)| + |\Phi_2(T_1)|) - |\Phi_1(T_1) \cup \Phi_2(T_1)|\right]\\
&= \mathbb{E}_\pi\left[|\Phi_1(T_1)| + |\Phi_2(T_1)| - |\mathsf{OPT}|\right] && \text{(because } T_1 \text{ is a maximal solution)}\\
&\leq 2\,\mathbb{E}_\pi\left[|T_1|\right] - |\mathsf{OPT}| && \text{(because } |T_1| \geq |\Phi_i(T_1)|\text{)}\\
&\leq 2\epsilon\,|\mathsf{OPT}|.
\end{aligned}$$

Now to prove Eq (B.3), we first bound $|\Phi_1(T_f)| + |\Phi_2(T_f)|$. It is at least

$$\begin{aligned}
&|\mathsf{OPT}| + \sum_{e \in \mathsf{OPT}} \mathbf{1}[e \in \mathsf{span}_1(T_f) \cap \mathsf{span}_2(T_f)] - \sum_{e \in \mathsf{OPT}} \mathbf{1}[e \notin (\mathsf{span}_1(T_f) \cup \mathsf{span}_2(T_f))]\\
&\geq |\mathsf{OPT}| + \sum_{e \in \mathsf{OPT}} \mathbf{1}[e \in T_f] - \sum_{e \in \mathsf{OPT}} \mathbf{1}[e \notin \mathsf{span}_1(T_f) \cup \mathsf{span}_2(T_f)]. \quad \text{(because } T_f \subseteq \mathsf{span}_i(T_f)\text{)}
\end{aligned}$$

Taking expectations and using Claim B.4.1,

$$\mathbb{E}_\pi[|\Phi_1(T_f)| + |\Phi_2(T_f)|] \geq |\mathsf{OPT}| - \left(\frac{1}{f} - 2\right)\mathbb{E}_\pi[|T_f \cap \mathsf{OPT}|] \tag{B.4}$$

Since $f \leq \frac{1}{2}$, we can use an upper bound on $\mathbb{E}_\pi[|T_f \cap \mathsf{OPT}|]$. Observe $T_1 \supseteq T_f$ is a maximal solution implying $|T_1| \geq |T_1 \cap \mathsf{OPT}| + \frac{1}{2}(|\mathsf{OPT}| - |T_1 \cap \mathsf{OPT}|) \geq \frac{1}{2}(|\mathsf{OPT}| + |T_f \cap \mathsf{OPT}|)$. Taking expectations,

$$\mathbb{E}_\pi[|T_f \cap \mathsf{OPT}|] \leq 2\,\mathbb{E}_\pi\left[|T_1| - \frac{1}{2}|\mathsf{OPT}|\right] \leq 2\,\epsilon\,|\mathsf{OPT}|. \quad \text{(because } \mathbb{E}_\pi[|T_1|] \leq \left(\tfrac{1}{2} + \epsilon\right)|\mathsf{OPT}|\text{)}$$

Combining this with Eq. (B.4) and Eq. (B.2) proves Eq. (B.3),

$$\begin{aligned}
\mathbb{E}_\pi\left[|\Phi_1(T_f^\pi) \cup \Phi_2(T_f^\pi)|\right] &= \mathbb{E}_\pi[|\Phi_1(T_f)| + |\Phi_2(T_f)|] - \mathbb{E}_\pi\left[|\Phi_1(T_f^\pi) \cap \Phi_2(T_f^\pi)|\right]\\
&\geq \left(1 - \frac{2\epsilon}{f} + 2\epsilon\right)\,|\mathsf{OPT}|.
\end{aligned}$$

Finally, using Eq. (B.4) and $|T_f| \geq |\Phi_i(T_f)|$, we also have $\mathbb{E}_\pi[|T_f^\pi|] \geq \frac{1}{2}\mathbb{E}_\pi[|\Phi_1(T_f)| + |\Phi_2(T_f)|] \geq \left(\frac{1}{2} - \left(\frac{1}{f} - 2\right)\epsilon\right)\,|\mathsf{OPT}|$. ■

For intuition, imagine the following claim for $f = \frac{1}{2}$, where it says that for a uniformly random order probability that e is in not in the span of T_f for either of the matroids is at most the probability e is selected by GREEDY into T_f.

Claim B.4.1 *Suppose* $\mathcal{G}(1) \leq \left(\frac{1}{2} + \epsilon\right) |\mathsf{OPT}|$ *for some* $\epsilon < \frac{1}{2}$ *and* T_f *is the output of* GREEDY *on* $E([1, mf)]$*, then*

$$\forall e \in \mathsf{OPT} \qquad \Pr_{\pi}[e \notin \Phi_1(T_f) \wedge e \notin \Phi_2(T_f)] \leq \left(\frac{1}{f} - 1\right) \Pr_{\pi}[e \in T_f].$$

Proof: Let us define the event $\mathcal{X} = \Big(e \notin \Phi_1(T_f) \wedge e \notin \Phi_2(T_f)\Big) \vee (e \in T_f)$. Consider the mapping g from permutations to permutations. If e occurs in the first f fraction of elements then $g(\pi) = \pi$. If not, then remove e and insert it uniformly at randomly at a position in $[1, mf]$. This induces a mapping from the set of all permutations on the ground elements to the set of permutations that have e in the first f fraction of elements. The important observation is that the set of permutations satisfying the event $\mathcal{X}$ still satisfy the event under the mapping g. We can conclude that $\Pr[\mathcal{X}] \leq \Pr[\mathcal{X} \mid e \in [1, mf]]$. Conditioned on the event that $e \in [1, fm]$, event $\mathcal{X}$ means $e \in T_f$. This is because if $e \notin \Phi_1(T_f) \wedge e \notin \Phi_2(T_f)$ and $e \in E[1, fm]$ then $T_f \cup e \in \mathcal{M}_1 \cap \mathcal{M}_2$. Thus, we can conclude that $\Pr[\mathcal{X}] \leq \Pr[e \in T_f \mid e \in [1, mf]] = \frac{1}{f} \Pr[e \in T_f]$. Moreover, since $\Big(e \notin \Phi_1(T_f) \wedge e \notin \Phi_2(T_f)\Big)$ and $(e \in T_f)$ are disjoint events, $\Pr[\mathcal{X}] = \Pr\left[\Big(e \notin \Phi_1(T_f) \wedge e \notin \Phi_2(T_f)\Big)\right] + \Pr[e \in T_f]$, which proves this claim. ■

Appendix C

Online Bin Packing Appendix

C.1 Omitted Proofs of Section 5.4 (Unit Movement Costs)

Here we provide proofs for our unit recourse upper and lower bounds.

First, we show that the optimal value of (LP_ε), restated below for ease of reference, is roughly $\alpha \approx 1.3871$, where we recall that α is such that $1 - 1/\alpha$ is a solution to the following equation

$$3 + \ln(1/2) = \ln(x) + 1/x. \tag{C.1}$$

Lemma 5.4.2 *The optimal value $\alpha_\varepsilon^\star$ of (LP_ε) satisfies $\alpha_\varepsilon^\star \in [\alpha - O(\varepsilon), \alpha + O(\varepsilon)]$.*

$$\begin{aligned}
&\text{minimize } \alpha_\varepsilon && && && (\text{LP}_\varepsilon)\\
&\text{s.t. } N_0 + \textstyle\sum_{x \in \mathcal{S}_\varepsilon} (1-x) \cdot N_x && \geq B - 1/B^c && && (\text{Vol}_\varepsilon)\\
&\quad N_0 + \textstyle\sum_{x \in \mathcal{S}_\varepsilon} N_x && \leq \alpha_\varepsilon \cdot B && && (\text{small}_\varepsilon)\\
&\quad N_0 + \textstyle\sum_{x \in \mathcal{S}_\varepsilon, x \leq t-\varepsilon} N_x + \left\lfloor \frac{B}{1-t} \right\rfloor && \leq \alpha_\varepsilon \cdot \left\lceil \frac{B}{1-t} \right\rceil && \forall t \in \mathcal{S}_\varepsilon && (\text{CR}_\varepsilon)\\
&\quad N_x \geq 0
\end{aligned}$$

Proof: We first modify (LP_ε) slightly to make it easier to work with—this will affect its optimal value only by $O(\varepsilon)$.

(i) Change the $B - 1/B^c$ term in the RHS of inequality (Vol_ε) to B, and remove the floor and ceiling in the inequalities (CR_ε). As $B \geq 1/\varepsilon$, this affects the optimal value by $O(\varepsilon)$.

(ii) Divide the inequalities through by B, and introduce new variables n_x for N_x/B, and

(iii) Replace $\alpha_\varepsilon - 1$ by a new variable α'_ε, and change the objective value to $\alpha'_\varepsilon + 1$.

This gives the LP (LPnew_ε), whose optimal value is $\alpha_\varepsilon^\star \pm O(\varepsilon)$. We will provide a feasible solution to this linear program and a feasible dual solution for its dual linear program (Dual_ε) whose objective values are $\alpha + O(\varepsilon)$ and $\alpha - O(\varepsilon)$, respectively. This proves the desired result.

Upper Bounding $\mathbf{OPT}(\mathbf{LPnew}_\varepsilon)$. We first give a solution that is nearly feasible, and then modify it to give a feasible solution with value at most $\alpha + O(\varepsilon)$.

$$
\begin{aligned}
&\text{min. } \alpha'_\varepsilon + 1 && && && (\text{LPnew}_\varepsilon)\\
&\quad n_0 + \sum_{x\in\mathcal{S}_\varepsilon} (1-x)\cdot n_x \geq 1 && && &&\\
&\quad n_0 + \sum_{x\in\mathcal{S}_\varepsilon} n_x - \alpha'_\varepsilon && \leq 1 && &&\\
&\quad n_0 + \sum_{x\in\mathcal{S}_\varepsilon, x\leq t-\varepsilon} n_x && \leq \frac{\alpha'_\varepsilon}{1-t} && \forall t\in\mathcal{S}_\varepsilon &\\
&\quad n_x \geq 0 && && &&
\end{aligned}
$$

$$
\begin{aligned}
&\text{max. } Z - q_0 + 1 && && && (\text{Dual}_\varepsilon)\\
&\quad q_0 + \sum_{t\in\mathcal{S}_\varepsilon} \frac{q_t}{1-t} && \leq 1 && && (\text{d1})\\
&\quad q_0 + \sum_{t\in\mathcal{S}_\varepsilon} q_t && \geq Z && && (\text{d2})\\
&\quad q_0 + \sum_{t\geq x+\varepsilon, t\in\mathcal{S}_\varepsilon} q_t \geq (1-x)\cdot Z && && \forall x\in\mathcal{S}_\varepsilon && (\text{d3})
\end{aligned}
$$

Figure C.1: The modified LP and its dual program

Let C denote $\alpha - 1$. Define $n_x := \int_{x-\varepsilon}^{x} \frac{C}{(1-y)^2}\, dy$ for all $x \in \mathcal{S}_\varepsilon$ and

$$n_0 := 1 - \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{C\cdot dy}{(1-y)} = 1 - C\ln\Big(\frac{1}{2}\Big) + C\ln\Big(1-\frac{1}{\alpha}\Big).$$

The first constraint of (LPnew$_\varepsilon$) is satisfied up to an additive $O(\varepsilon)$:

$$
\begin{aligned}
n_0 + \sum_x (1-x)n_x &= 1 - \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{C}{(1-y)}\, dy + \sum_{x\in\mathcal{S}_\varepsilon} (1-x)\int_{x-\varepsilon}^{x} \frac{C}{(1-y)^2}\, dy\\
&\geq 1 - \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{C}{(1-y)}\, dy + \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{C(1-y)}{(1-y)^2}\, dy - O(\varepsilon)\\
&= 1 - O(\varepsilon).
\end{aligned}
$$

Next, the second constraint of (LPnew$_\varepsilon$).

$$
\begin{aligned}
n_0 + \sum_{x\in\mathcal{S}_\varepsilon} n_x &= 1 - \int_{1/2}^{\frac{1}{\alpha}} \frac{C\cdot dy}{1-y} + \sum_{x\in\mathcal{S}_\varepsilon} \int_{x-\varepsilon}^{x} \frac{C\cdot dy}{(1-y)^2}\\
&= 1 - \int_{1/2}^{\frac{1}{\alpha}} \frac{C\cdot dy}{1-y} + \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{C\cdot dy}{(1-y)^2}\\
&= 1 + C\Big(-\ln(1/2) + \ln\Big(1-\frac{1}{\alpha}\Big) - 2 + \frac{1}{1-\frac{1}{\alpha}}\Big)\\
&= 1 + C\\
&= \alpha,
\end{aligned}
$$

where the penultimate step follows by (C.1), and the last step uses $C = \alpha - 1$. Performing a similar

calculation for the last set of constraints in (LPnew$_\varepsilon$), we get

$$\begin{aligned}
n_0 + \sum_{x\in\mathcal{S}_\varepsilon, x<t-\varepsilon} n_x &= n_0 + \sum_{x\in\mathcal{S}_\varepsilon} n_x - \sum_{x\in\mathcal{S}_\varepsilon, x\geq t-\varepsilon} n_x \\
&= \alpha - \sum_{x\geq t-\varepsilon, x\in\mathcal{S}_\varepsilon} \int_{x-\varepsilon}^{x} \frac{C}{(1-y)^2}\,dy \\
&\leq \alpha - \int_{t}^{\frac{1}{\alpha}} \frac{C}{(1-y)^2}\,dy + O(\varepsilon) \\
&= \alpha - C\Big(\frac{\alpha}{\alpha-1} - \frac{1}{1-t}\Big) + O(\varepsilon) \\
&= \frac{\alpha-1}{1-t} + O(\varepsilon),
\end{aligned}$$

where the second equality follows from the previous sequence of calculations. To satisfy the constraints of (LPnew$_\varepsilon$), we increase n_0 to $n_0 + O(\varepsilon)$ and set α'_ε to $\alpha - 1 + O(\varepsilon)$. Since t is always $\geq 1/2$, this will also satisfy the last set of constraints. Since the optimal value of (LPnew$_\varepsilon$) is $\leq \alpha - 1 + O(\varepsilon)$, which implies that $\alpha^\star_\varepsilon \leq \alpha - 1$, since we had subtracted 1 from the objective function when we constructed (LPnew$_\varepsilon$) from (LP$_\varepsilon$)).

Lower Bounding OPT(LPnew$_\varepsilon$). As with our upper bound on OPT(LPnew$_\varepsilon$), we start with a nearly-feasible dual solution to (Dual$_\varepsilon$) and later modify it to obtain a feasible solution. Set $Z = \alpha(\alpha-1)$, $q_0 = (\alpha-1)^2$, $q_{\frac{1}{2}+\varepsilon} = \alpha(\alpha-1)/2$, and $q_t = \alpha(\alpha-1)\varepsilon$ for all $t \in \mathcal{S}_\varepsilon$ with $t \geq \frac{1}{2}+2\varepsilon$. The objective value of (Dual$_\varepsilon$) with respect to this solution is exactly $\alpha(\alpha-1) - (\alpha-1)^2 + 1 = \alpha$. We will now show that it (almost) satisfies the constraints. For sake of brevity, we do not explicitly write that variable t takes values in $\mathcal{S}_\varepsilon$ in the limits for the sums below. First, consider constraint (d1):

$$\begin{aligned}
q_0 + \sum_{t=\frac{1}{2}+\varepsilon}^{\frac{1}{\alpha}} \frac{q_t}{1-t} &= q_0 + \frac{q_{\frac{1}{2}+\varepsilon}}{\frac{1}{2}-\varepsilon} + \sum_{t>\frac{1}{2}+\varepsilon}^{\frac{1}{\alpha}} \frac{q_t}{1-t} \\
&\leq q_0 + 2q_{\frac{1}{2}+\varepsilon} + \sum_{t>\frac{1}{2}+\varepsilon}^{\frac{1}{\alpha}} \frac{q_t}{1-t} \\
&\leq (\alpha-1)^2 + \alpha(\alpha-1) + \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{\alpha(\alpha-1)\,dx}{1-x} \\
&= (\alpha-1)^2 + \alpha(\alpha-1) + \alpha(\alpha-1)\big(\ln\left(\frac{1}{2}\right) - \ln\left(1-\frac{1}{\alpha}\right)\big) \\
&= (\alpha-1)^2 + \alpha(\alpha-1) + \alpha(\alpha-1)\Big(\frac{3-2\alpha}{\alpha-1}\Big) \qquad = 1
\end{aligned}$$

where we used Equation (C.1), which follows from the definition of α, in the penultimate equation.

Next, consider constraint (d2).

$$\begin{aligned} q_0 + \sum_{t=\frac{1}{2}+\varepsilon}^{\frac{1}{\alpha}} q_t &= q_0 + q_{\frac{1}{2}+\varepsilon} + \sum_{t>\frac{1}{2}+\varepsilon}^{\frac{1}{\alpha}} q_t \\ &\geq (\alpha-1)^2 + \frac{1}{2}\alpha(\alpha-1) + \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \alpha(\alpha-1)\,dx - O(\varepsilon) \\ &= (\alpha-1)^2 + \frac{1}{2}\alpha(\alpha-1) + \alpha(\alpha-1)\Big(\frac{1}{\alpha} - \frac{1}{2}\Big) - O(\varepsilon) \qquad = Z - O(\varepsilon) \end{aligned}$$

Finally, consider constraint (d3) for any $x \in \mathcal{S}_\varepsilon$:

$$\begin{aligned} q_0 + \sum_{t \geq x+\varepsilon, t \in \mathcal{S}_\varepsilon} q_t &\geq (\alpha-1)^2 + \int_{x+\varepsilon}^{\frac{1}{\alpha}} \alpha(\alpha-1)\,dx - O(\varepsilon) \\ &= (\alpha-1)^2 + \Big(\frac{1}{\alpha} - x - \varepsilon\Big)\alpha(\alpha-1) - O(\varepsilon) \\ &= (\alpha-1)\Big(\alpha - 1 + \Big(\frac{1}{\alpha} - x - \varepsilon\Big)\cdot\alpha\Big) - O(\varepsilon) \\ &= Z \cdot (1 - x - \varepsilon) - O(\varepsilon). \end{aligned}$$

Finally, increase q_0 to $q_0 + O(\varepsilon)$ to ensure that constraints (d2) and (d3) are satisfied. This is now a feasible solution to (Dual$_\varepsilon$) with objective value $Z - q_o + 1 - O(\varepsilon) = \alpha - O(\varepsilon)$. Hence the optimal value $\alpha^\star_\varepsilon$ for the LP (LPnew$_\varepsilon$) is at least $\alpha - O(\varepsilon)$. ■

C.1.1 Proof of the Lower Bound

We now prove that $\alpha^\star_\varepsilon$, the optimal value of (LP$_\varepsilon$), is such that any algorithm with a.c.r below this $\alpha^\star_\varepsilon$ must have either polynomial additive term or recourse (or both).

Lemma 5.4.3 *For all $\varepsilon > 0$ and $\frac{1}{2} > \delta > 0$, if $\alpha^\star_\varepsilon$ is the optimal value of* (LP$_\varepsilon$)*, then any fully-dynamic bin packing algorithm $\mathcal{A}$ with a.c.r $\alpha^\star_\varepsilon - \varepsilon$ and additive term $o(\varepsilon^2 \cdot n^\delta)$ has recourse $\Omega(\varepsilon^2 \cdot n^{1-\delta})$ under unit movement costs.*

Proof: For any $x \in \mathcal{S}_\varepsilon \cup \{0\}$, again define N_x as the number of bins with free space in the range $[x, x+\varepsilon)$ when $\mathcal{A}$ faces input $\mathcal{I}_s$. Inequality (Vol$_\varepsilon$) is satisfied for the same reason as above. Recall that $B = \Theta(n^\delta)$. As $\mathcal{A}$ is $(\alpha^\star_\varepsilon - \Omega(\varepsilon))$-asymptotically competitive with additive term $o(\varepsilon \cdot n^\delta)$, i.e., $o(\varepsilon \cdot B)$, and $OPT(\mathcal{I}_s) = B$, we have $N_0 + \sum_{x \in \mathcal{S}_\varepsilon} N_x \leq (\alpha^\star_\varepsilon - \Omega(\varepsilon) + o(\varepsilon)) \cdot B \leq \alpha^\star_\varepsilon \cdot B$. That is, the N_x's satisfy constraint (small$_\varepsilon$) with $\alpha_\varepsilon = \alpha^\star_\varepsilon$.

We now claim that there exists a $\ell \in \mathcal{S}_\varepsilon$ such that

$$N_0 + \sum_{x \in \mathcal{S}_\varepsilon, x \leq \ell - \varepsilon} N_x + \Big\lfloor \frac{B}{1-\ell} \Big\rfloor \geq \alpha^\star_\varepsilon \cdot \Big\lceil \frac{B}{1-\ell} \Big\rceil \tag{C.2}$$

holds (notice the opposite inequality sign compared to constraint (CR$_\varepsilon$)). Suppose not. Then the quantities N_0, N_x for $x \in \mathcal{S}_\varepsilon$, and $\alpha^\star_\varepsilon$ strictly satisfy the constraints (CR$_\varepsilon$). If they also strictly

satisfy the constraint (small$_\varepsilon$), then we can maintain feasibility and slightly reduce $\alpha^\star_\varepsilon$, which contradicts the definition of $\alpha^\star_\varepsilon$. Therefore assume that constraint (small$_\varepsilon$) is satisfied with equality. Now two cases arise: (i) All but one variable among $\{N_0\} \cup \{N_x \mid x \in \mathcal{S}_\varepsilon\}$ are zero. If this variable is N_0, then tightness of (small$_\varepsilon$) implies that $N_0 = \alpha^\star_\varepsilon B$. But then we satisfy (Vol$_\varepsilon$) with slack, and so, we can reduce N_0 slightly while maintaining feasibility. Now we satisfy all the constraints strictly, and so, we can reduce $\alpha^\star_\varepsilon$, a contradiction. Suppose this variable happens to be N_x, where $x \in \mathcal{S}_\varepsilon$. So, $N_x = \alpha^\star_\varepsilon B$. We will show later in Theorem 5.4.2 that $\alpha^\star_\varepsilon \leq 1.4$. Since $(1-x) \leq 1/2$, it follows that $(1-x)N_x \leq 0.7B$, and so we satisfy (Vol$_\varepsilon$) with slack. We again get a contradiction as argued for the case when N_0 was non-zero, (ii) There are at least two non-zero variables among $\{N_0\} \cup \{N_x \mid x \in \mathcal{S}_\varepsilon\}$ – let these be N_{x_1} and N_{x_2} with $x_1 < x_2$ (we are allowing x_1 to be 0). Now consider a new solution which keeps all variables N_x unchanged except for changing N_{x_1} to $N_{x_1} + \frac{\eta}{1-x}$, and N_{x_2} to $N_{x_2} - \frac{\eta}{1-x}$, where η is a small enough positive constant (so that we continue to satisfy the constraints (CR$_\varepsilon$) strictly). The LHS of (Vol$_\varepsilon$) does not change, and so we continue to satisfy this. However LHS of (small$_\varepsilon$) decreases strictly. Again, this allows us to reduce $\alpha^\star_\varepsilon$ slightly, which is a contradiction. Thus, there must exist a ℓ which satisfies (C.2). We fix such a ℓ for the rest of the proof.

Let $\mathcal{B}$ denote the bins which have less than ℓ free space. So, $|\mathcal{B}| = N_0 + \sum_{x \in \mathcal{N}_\varepsilon : x \leq \ell - \varepsilon} N_x$. Now, we insert $\lfloor \frac{B}{1-\ell-\varepsilon} \rfloor$ items of size $\ell + \varepsilon$. (It is possible that $\ell = 1/\alpha$, and so $\ell + \varepsilon \notin \mathcal{S}_\varepsilon$, but this is still a valid instance). We claim that the algorithm must move at least ε volume of small items from at least εB bins in $\mathcal{B}$. Suppose not. Then the large items of size $\ell + \varepsilon$ can be placed in at most εB bins in $\mathcal{B}$. Therefore, the total number of bins needed for $\mathcal{I}_\ell$ is at least $N_0 + \sum_{x \in \mathcal{N}_\varepsilon : x \leq \ell - \varepsilon} N_x - \varepsilon B + \left\lfloor \frac{B}{1-\ell} \right\rfloor$, which by inequality (C.2), is at least $(\alpha^\star_\varepsilon - O(\varepsilon)) \cdot OPT(\mathcal{I}_{\ell+\varepsilon})$, because $OPT(\mathcal{I}_{\ell+\varepsilon}) = \left\lceil \frac{B}{1-\ell-\varepsilon} \right\rceil = \left\lceil \frac{B}{1-\ell} \right\rceil + O(\varepsilon B)$. But we know that $\mathcal{A}$ is $(\alpha^\star_\varepsilon - \Omega(\varepsilon))$-asymptotically competitive with additive term $o(\varepsilon \cdot n^\delta)$ (which is $o(\varepsilon \cdot OPT(\mathcal{I}_{\ell+\varepsilon}))$. So it should use at most $(\alpha^\star_\varepsilon - \Omega(\varepsilon) + o(\varepsilon)) \cdot OPT(\mathcal{I}_{\ell+\varepsilon})$ bins, which is a contradiction. Since each small item has size $1/B^c$, the total number of items moved by the algorithm is at least $\varepsilon^2 B / B^c$. This is $\Omega(\varepsilon \cdot n^{1-\delta})$, because $\varepsilon \geq 1/B$, and $B^c = \Theta(n^{1-\delta})$. ■

C.1.2 Matching Algorithmic Results

We now address the omitted proofs of our matching upper bound.

LP$_\varepsilon$ as a Factor-Revealing LP

We now present the omitted proofs allowing us to use (LP$_\varepsilon$) to upper bound our algorithm's a.c.r. We start by showing that an optimal solution to (LP$_\varepsilon$) induces a packing of the small items which can be trivially extended (i.e., without moving any items) to an $(\alpha^\star_\varepsilon + O(\varepsilon))$-competitive packing of any number of ℓ-sized items, for any $\ell > 1/2$.

Lemma 5.4.4 (Huge Items of Same Size) *Any α_ε-feasible packing of small items of $\mathcal{I}_s$ induces an α_ε-competitive packing for all extensions $\mathcal{I}^k_\ell$ of $\mathcal{I}_s$ with $\ell > 1/2$ and $k \in \mathbb{N}$.*

Proof: Fix ℓ and k. Let N_x be the bins with exactly free space in the packing of $\mathcal{I}_s$ and let $N = \sum_{x|x\geq\ell, x\in\mathcal{S}_\varepsilon} N_x$ be the bins with at least ℓ free space; if $\ell \geq 1/\alpha$, then $N = 0$. Let $N' = \sum_{x|x\leq\ell-\varepsilon, x\in\mathcal{S}_\varepsilon} N_x$. Our algorithm first packs the size-ℓ items in the N bins of the packing before using new bins, and hence uses $N' + \max(N, k)$ bins. If $k \leq N$, we are done because of the constraint (small$_\varepsilon$), so assume $k \geq N$. A volume argument bounds the number of bins in the optimal solution for $\mathcal{I}_\ell^k$:

$$\text{OPT}(\mathcal{I}_\ell^k) \geq \begin{cases} k & \text{if } k(1-\ell) \geq B \\ k + \big(B - k(1-\ell)\big) & \text{else.} \end{cases}$$

We now consider two cases:

- $k(1-\ell) \geq B$: Using constraint (CR$_\varepsilon$), the number of bins used by our algorithm is

$$N' + k \leq \tfrac{\alpha_\varepsilon B}{1-\ell} + O(\varepsilon B) + \big(k - \tfrac{B}{1-\ell}\big) \leq (\alpha_\varepsilon + O(\varepsilon))k.$$

- $k(1-\ell) < B$: Since k lies between N and $\frac{B}{1-\ell}$, we can write it as a convex combination $\frac{\lambda_1 B}{1-\ell} + \lambda_2 N$, where $\lambda_1 + \lambda_2 = 1, \lambda_1, \lambda_2 \geq 0$. We can rewrite constraints (small$_\varepsilon$) and (CR$_\varepsilon$) as

$$N' + \tfrac{B}{1-\ell} \leq \tfrac{\alpha_\varepsilon B}{1-\ell} + O(\varepsilon B) \quad \text{and} \quad N' + N \leq \alpha_\varepsilon B.$$

  Combining them with the same multipliers λ_1, λ_2, we see that $N' + k$ is at most

$$\alpha_\varepsilon \left(\tfrac{\lambda_1 B}{1-\ell} + \lambda_2 B\right) + O(\varepsilon B) = \alpha_\varepsilon \left(B + \tfrac{\lambda_1 \ell B}{1-\ell}\right) + O(\varepsilon B) \leq \alpha_\varepsilon \left(B + \ell k\right) + O(\varepsilon B).$$

  The desired result follows because $B + \ell k = k + (B - k(1-\ell))$ and B is a lower bound on OPT$(\mathcal{I}_\ell^k)$.

■

We now proceed to provide a linear-time algorithm which for any fixed ϵ, given an input $\mathcal{I}$ produces a packing into $(1 + O(\varepsilon)) \cdot OPT(\mathcal{I})$ bins such that in almost all bins large items occupy either no space or more than half the bin.

Observation 5.4.5 *For any input $\mathcal{I}$ made up of solely large items and function $f(\cdot)$, a packing of $\mathcal{I}$ using at most $(1+\varepsilon)\cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ bins has all but at most $2\varepsilon \cdot OPT(\mathcal{I}) + 2f(\varepsilon^{-1}) + 3$ of its bins containing either no large items or being more than half filled by large items.*

Proof: Suppose this packing uses $N \leq (1+\varepsilon)\cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ bins of which $B \geq 2\varepsilon \cdot OPT(\mathcal{I}) + 2f(\varepsilon^{-1}) + 4$ are "bad" bins – bins with some $v \in (0, 1/2]$ volume taken up by large items. We show that this packing can be improved to require fewer than $OPT(\mathcal{I})$, which would lead to a contradiction. Indeed, repeatedly combining the contents of any two bad bins until no two such bins remain would decrease the number of bins by one and the number of bad bins by at most two for each combination (so this process can be repeated at least $\lfloor B/2 \rfloor \geq \varepsilon \cdot OPT(\mathcal{I}) + f(\varepsilon^{-1}) + 1$), and so would result in a new packing of the $\mathcal{I}$ using at most $N - \lfloor B/2 \rfloor < OPT(\mathcal{I})$ bins. ■

Theorem 5.4.6 *An α_ε-feasible packing of the small items of an instance $\mathcal{I}$ can be extended into a packing of all of $\mathcal{I}$ using at most $(\alpha_\epsilon + O(\varepsilon)) \cdot OPT(\mathcal{I}) + O(\varepsilon^{-2})$ bins in linear time for any fixed ϵ.*

Proof: We run the linear-time algorithm of De La Vega and Lueker [46] to compute a packing of the large items of $\mathcal{I}$ into at most $(1+\varepsilon)\cdot OPT(\mathcal{I})+O(\varepsilon^{-2})$ many bins. By Observation 5.4.5, all but at most $2\varepsilon\cdot OPT(\mathcal{I})+O(\varepsilon^{-2})$ of these bins have at most $\frac{1}{2}$ volume occupied by small items. We use these bins in our packing of $\mathcal{I}'$. For the remaining bins we "glue" all large items occupying the same bin into a single *huge* item. Note that any packing of $\mathcal{I}'$ trivially induces a similar packing of the as-of-yet unpacked large items of $\mathcal{I}$ with the same number of bins (simply pack large items glued together in the place of their induced glued item). Moreover, by construction all large items of $\mathcal{I}'$ are huge (i.e., have size greater than $1/2$), and clearly $OPT(\mathcal{I}') \leq (1+O(\varepsilon))\cdot OPT(\mathcal{I})+O(\varepsilon^{-2})$, as $\mathcal{I}'$ can be packed using this many bins. As the free space in all bins is an integer multiple of ϵ, we can round the huge items' sizes to integer multiples of ϵ and obtain a packing with the same number of bins for $\mathcal{I}'$. Such a rounding allows us to bucket sort the huge items of $\mathcal{I}'$ (and bins) in time $O(n/\epsilon)$. All the above steps take $O_\varepsilon(n)$ time. It remains to address the obtained packing's approximation ratio.

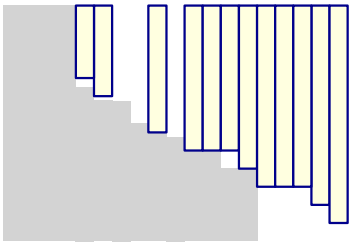

Figure C.2: A greedy packing of instance $\mathcal{I}'$. Large items are packed "on top" of the small items (in grey).

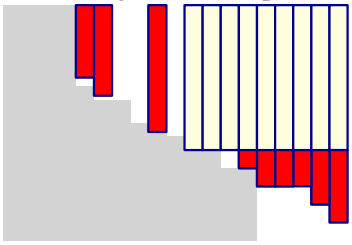

Figure C.3: Packing of instance $\mathcal{I}'^k_\ell$ obtained from $\mathcal{I}'$ by removing parts of huge items $\mathcal{I}'^k_\ell$ (in red).

Figure C.4: A packing of instance $\mathcal{I}'$ and the instance $\mathcal{I}'^k_\ell$ obtained from $\mathcal{I}'$.

In order to upper bound the number of bins used to pack $\mathcal{I}'$ (and therefore $\mathcal{I}$), we create a new instance of the form $\mathcal{I}'^k_\ell$ for some k and $\ell > 1/2$. Specifically, if we sort the bins containing small items in decreasing order of free space, we remove all large items packed in a bin with f free space such that some bin with at least f free space contains no large item of $\mathcal{I}$. Let the smallest remaining large item size be ℓ. We decrease the size of all k remaining larger items to ℓ, yielding the instance $\mathcal{I}'^k_\ell$ packed on top of the curve using the same number of bins as our packing of $\mathcal{I}$ (see Figure C.3). By Lemma 5.4.4, ignoring the additive $O(\epsilon^{-2})$ term due to the packing of the large items, the packing of $\mathcal{I}'$ on top of the small items – and hence the packing we obtain for $\mathcal{I}$ – uses at most $\alpha_\varepsilon \cdot OPT(\mathcal{I}'^k_\ell) \leq \alpha_\varepsilon \cdot OPT(\mathcal{I}') \leq \alpha_\varepsilon \cdot (1+O(\epsilon)) \cdot OPT(\mathcal{I})$ bins. ■

Theorem 5.4.6 immediately gives us amortized recourse bounds. Extending it slightly allows us to obtain worst-case recourse bounds, as follows: replace the static $(1+\epsilon)$-a.c.r algorithm by a

dynamic algorithm and round the huge items to sizes which are multiples of ϵ. Doing this naively yields an $(\alpha + O(\epsilon))$ a.c.r, with worst-case recourse bounds that are at most $O(\epsilon^{-3})$ times the recourse bounds of a fully-dynamic $(1+\epsilon)$-a.c.r bin packing algorithm for items of size at least ϵ. Using the worst-case recourse bounds from Berndt et al. [25, Theorem 9], we get an $\tilde{O}(\epsilon^{-6})$ worst-case recourse bound.

The following theorem improves this worst-case recourse bound to being only an $O(\epsilon^{-1})$-times worse, instead of the naive $O(\epsilon^{-3})$ times worse. This, combined with Berndt et al. [25, Theorem 9], gives an improved $\tilde{O}(\epsilon^{-4})$ worst-case recourse bound.

Theorem 5.4.7 *For a dynamic instance $\mathcal{I}_t$, given a fully-dynamic $(1+\epsilon)$-a.c.r algorithm with additive term $f(\epsilon^{-1})$ for items of size greater than $1/4$ in $\mathcal{I}_t$, and a fully-dynamic α_ε-feasible packing of its small items, one can maintain a packing with a.c.r $(\alpha_\epsilon + O(\varepsilon))$ with additive term $O(f(\epsilon^{-1}))$. This can be done using worst-case recourse*

1. *$O(\varepsilon^{-1})$ per item move in the near-optimal fully-dynamic packing of the items of size $> 1/4$,*
2. *$O(\epsilon^{-1})$ per insertion or deletion of medium items, and*
3. *$O(1)$ per item move in the α_ϵ-feasible packing of the small items.*

Proof: The idea here is similar to the proof of Theorem 5.4.6, and we mainly discuss the recourse bound.

By the theorem's hypothesis, we have a $(1+\epsilon)$-a.c.r packing of the subinstance made up of the large items of size greater than $1/4$. We "glue" items in bins which are at least half full into single huge items, yielding an instance $\mathcal{I}'$ with $OPT(\mathcal{I}') \leq (1+O(\varepsilon)) \cdot OPT(\mathcal{I}) + O(f(\varepsilon^{-1}))$, and which by Observation 5.4.5 has at most $2\varepsilon \cdot OPT(\mathcal{I}) + O(f(\varepsilon^{-1}))$ non-huge items. We pack these items in individual bins. We pack $\mathcal{I}'$ greedily on top of the curve, by packing the huge items in order of increasing huge item size, according to FIRSTFIT, with the bins sorted in increasing order of free space (see Figure C.2). As in the linear-time algorithm of Theorem 5.4.6, rounding the huge items' sizes to integer products of ϵ does not increase the number of bins used to pack $\mathcal{I}'$. For Theorem 5.4.6 this allowed us to pack the huge items in linear time. This rounding also proves useful in obtaining low worst-case recourse, as follows.

First, consider only the small items and the large items of size greater than $1/4$. We dynamically pack the former into some α_ϵ-feasible packing, and the large items using the fully-dynamic $(1+\epsilon)$-a.c.r algorithm of the theorem's statement. Whenever a bin is changed in the packing of the large items of size greater than $1/4$ (an item added/removed by the dynamic packing algorithm), we either changed a bin which was less than half full (in which case we move its $O(1)$ items in our packing of $\mathcal{I}_t$), or we create a new item in the dynamic instance $\mathcal{I}'_t$. Now, we can dynamically keep the huge items of $\mathcal{I}'_t$ packed as though inserted in sorted FIRSTFIT order as above as follows: whenever a huge item is added, we add it in the bin "after" all items of the same size, possibly removing an item of larger size from this bin, which we then re-insert (deletion is symmetric). As the number of different huge item sizes is $O(\varepsilon^{-1})$, we move at most $O(\varepsilon^{-1})$ huge items, and so at most $O(\varepsilon^{-1})$ large items of $\mathcal{I}_t$ (as these large items have size greater than $1/4$, and so the huge items correspond to at most three such items). Finally, we now discuss how to pack medium items (items of size in the range $(\epsilon, 1/4]$).

We strive to keep the medium-sized items on top of our packing for the large and small items so as to guarantee that the bins in the prefix of all but the last bin (sorted by time of opening) which

contain a medium-sized item are all at least 3/4 full if we ignore the fact that small item groups and huge items have their sizes rounded to products of ϵ. By a volume argument (accounting for above-mentioned rounding), if we open a new bin, our packing has a.c.r $4/3 + O(\varepsilon) < \alpha + (\varepsilon)$. If no new bin is opened, we can safely ignore the medium-sized items and compare the number of bins we open against an easier instance which does not contain these medium-sized items, which as we shall see later yields an a.c.r of $\alpha + O(\epsilon)$. We now discuss details of dynamically packing medium-sized items in this way.

Insertion of a medium-sized item is trivial using no recourse, by adding it to the first bin which is less than $3/4$ full and accommodates this new item (or opening a new bin if necessary). Removal of a medium-sized item is not much harder. Upon the removal of some or all medium-sized items from some bin B, we take the last medium items (according to the bins' order) and reinsert them into B until B is at least $3/4$ full or until it is the last bin with medium items. As medium items have size in the range $(\varepsilon, 1/4]$, this requires $O(\epsilon^{-1})$ worst-case recourse. Similarly, a change in the packing of small items can only increase or decrease the volume used in a bin by $O(\varepsilon)$. Such an insertion can be addressed by removing at most $O(1)$ items from the bin until it does not overflow, and then reinserting them (in the case of an insertion of small items into a bin), or by inserting some last $O(1)$ medium items into this bin in the case of deletion. It remains to address changes due to updates in the packing of large items (which are grouped into huge items).

Recall that all the huge items have their sizes rounded up to products of ϵ. This rounding allows us to obtain a packing as in Figure C.2 by only repacking $O(\epsilon^{-1})$ such huge items following an update to the packing of the large items, if we ignore medium items. Now, consider the medium items that are displaced due to such a move. Following a removal of a huge item, a huge item of larger size may replace the removed item, and this huge item may be replaced in turn by a larger huge item, and so on for the ϵ^{-1} different huge item sizes. We address potential overflowing of these bins by removing the minimum number of medium items to guarantee these bins do not overflow and repacking them. As each medium item has size greater than ϵ, we remove at most one such item per huge item size class; that is, $O(\epsilon^{-1})$ such items. Similarly, the removal of a huge item potentially causes a larger huge item to take its place, and so on for $O(\epsilon^{-1})$ size classes, until no next-sized huge item exists or the next huge item does not fit. To avoid overfilling of these bins, we move at most one medium item per size class (as the medium items have size greater than ϵ). These $O(\epsilon^{-1})$ are repacked as above. The last bin affected may find that it has too few medium items and is less than $3/4$ full; we address this using $O(\epsilon^{-1})$ worst-case recourse as in our solution for deletion of medium items in a bin.

To summarize, our worst-case recourse is $O(\varepsilon^{-1})$ per item move in the $(1+\varepsilon)$-a.c.r dynamic packing of the large items, $O(\epsilon^{-1})$ per addition or deletion of a medium item and $O(1)$ per item move in the packing of small items. It remains to bound the obtained a.c.r of our packing obtained by extending the α_ε-feasible packing of the small items.

Similarly to the proof of Theorem 5.4.6, the a.c.r of this algorithm is at most $\alpha + O(\epsilon)$ if no bins are opened due to the medium items. On the other hand, as argued before, this algorithm's a.c.r is at most $4/3 < \alpha + O(\epsilon)$ if such bins are opened. Finally, we observe that the additive term is at most $O(f(\epsilon^{-1}))$, incurred by the packing of the large items and the number of non-huge items stored in designated bins. ■

Dealing With Small Items: "Fitting a Curve"

We now consider the problem of packing ε-small items according to an approximately-optimal solution of (LP$_\varepsilon$), which we abstract thus:

[Bin curve-fitting] Given a list of bin sizes $0 \leq b_0 \leq b_1 \leq \ldots, b_K \leq 1$ and relative frequencies $f_0, f_1, f_2, \ldots, f_K$, such that $f_x \geq 0$ and $\sum_{x=0}^{K} f_x = 1$, an algorithm for the *bin curve-fitting problem* must pack a set of m of items with sizes $s_1, \ldots, s_m \leq 1$ into a minimal number of bins N such that for every $x \in [0, K]$ the number of bins of size b_x that are used by this packing lie in $\{\lfloor N \cdot f_x \rfloor, \lceil N \cdot f_x \rceil\}$.

If we have $K = 0$ with $b_0 = 1$ and $f_0 = 1$, we get standard BIN PACKING. We want to solve the problem only for (most of the) small items, in the fully- dynamic setting. We consider the special case with relative frequencies f_x being multiples of $1/T$, for $T \in \mathbb{Z}$; e.g., $T = O(\varepsilon^{-1})$. Our algorithm maintains bins in increasing sorted order of item sizes. The number of bins is always an integer product of T. Consecutive bins are aggregated into *clumps* of exactly T bins each, and clumps aggregated into $\Theta(1/\varepsilon)$ *buckets* each. Formally, each clump has T bins, with $f_x \cdot T \in \mathbb{N}$ bins of size $\approx b_x$ for $x = 0, \ldots, K$. The bins in a clump are ordered according to their target b_x, so each clump looks like a curve. Each bucket except the last consists of some $s \in [1/\varepsilon, 3/\varepsilon]$ consecutive clumps (the last bucket may have fewer than $1/\epsilon$ clumps). See Figure C.5. For each bucket, all bins except those in the last clump are full to within additive ε of their target size.

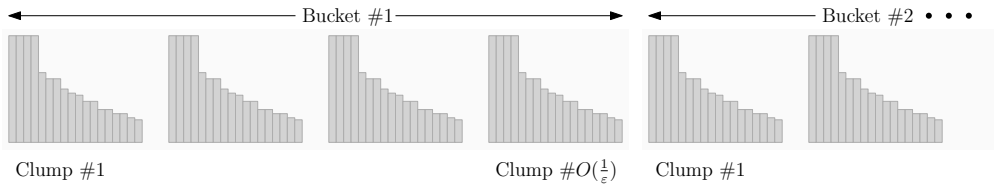

Figure C.5: Buckets have $O(1/\varepsilon)$ clumps, clumps have T bins.

Inserting an item adds it to the correct bin according to its size. If the bin size becomes larger than the target size for the bin, the largest item overflows into the next bin, and so on. Clearly this maintains the invariant that we are within an additive ε of the target size. We perform $O(T/\epsilon)$ moves in the same bucket; if we overflow from the last bin in the last clump of the bucket, we add a new clump of T new bins to this bucket, etc. If a bucket contains too many clumps, it splits into two buckets, at no movement cost. An analogous (reverse) process happens for deletes. Loosely, the process maintains that on average the bins are full to within $O(\varepsilon)$ of the target fullness – one loss of ε because each bin may have ε space, and another because an $O(\varepsilon)$ fraction of bins have no guarantees whatsoever.

As observed in §5.4, the above approach approach solves bin curve-fitting using $O(T/\varepsilon) = O(\varepsilon^{-2})$ worst-case recourse, provided all items have size at most ε (as in our case). provided all freq We now relate this process to the value of LP$_\varepsilon$. We first show that setting $T = O(1/\varepsilon)$ and restricting to frequencies to multiples of ε does not hurt us. Indeed, for us, $b_0 = 1$, and $b_x = (1-x)$ for $x \in \mathcal{S}_\varepsilon$. Since (LP$_\varepsilon$) depends on the total volume B of small items, and f_x may change if B changes, it is convenient to work with the normalized LP (LPnew$_\varepsilon$). Now n_x can be interpreted as just being proportional to number of bins of size b_x, and we can define $f_x = n_x / \sum_x n_x$. However,

we also need each f_x to be integer multiples of $1/T$ for some integer $T = O(1/\varepsilon)$. We achieve this by slightly modifying the LP solution (which requires a somewhat careful proof).

Lemma 5.4.8 (Multiples of ε) *For any optimal solution $\{n_x\}$ to* (LPnew$_\varepsilon$) *with objective value α_ε, we can construct in linear time a solution $\{\tilde{n}_x\} \subseteq \varepsilon \cdot \mathbb{N}$ with objective value $\alpha_\varepsilon + O(\varepsilon)$.*

Proof: For sake of brevity, let $\mathcal{S}'_\varepsilon := \mathcal{S}_\varepsilon \cup \{0\}$. Consider the indices $x \in \mathcal{S}'_\varepsilon$ in increasing order, and modify n_x in this order. Let $\Delta_x := \sum_{x' \in \mathcal{S}'_\varepsilon : x' < x} n_{x'}$, and let $\tilde{\Delta}_x$ be the analogous expression for $\tilde{n}_x$. We maintain the invariant that $|\Delta_x - \tilde{\Delta}_x| \leq \varepsilon$, which is trivially true for the base case $x = 0$. Inductively, suppose it is true for x. If $\Delta_x > \tilde{\Delta}_x$, define $\tilde{n}_x$ be n_x rounded up to the nearest multiple of ε, otherwise it is rounded down; this maintains the invariant. If we add $O(\varepsilon)$ to the old α'_ε value, this easily satisfies the second and third set of constraints, since our rounding procedure ensures that the prefix sums are maintained up to additive ε, and $t \in [0, \frac{1}{2}]$. Checking this for the first constraint turns out to be more subtle. We claim that

$$\tilde{n}_0 + \sum_{x \in \mathcal{S}_\varepsilon} (1 - x)\tilde{n}_x \geq 1 - O(\varepsilon).$$

For an element $x \in \mathcal{S}'_\varepsilon$, let Δn_x denote $n_x - \tilde{n}_x$. It is enough to show that $|\sum_{x \in \mathcal{S}'_\varepsilon} x \cdot \Delta n_x| \leq O(\varepsilon)$. Indeed,

$$\begin{aligned}\tilde{n}_0 + \sum_{x \in \mathcal{S}_\varepsilon} (1 - x)\tilde{n}_x &= n_0 + \sum_{x \in \mathcal{S}_\varepsilon} (1 - x)n_x - \Delta n_0 - \sum_{x \in \mathcal{S}_\varepsilon} (1 - x)\Delta n_x \\ &\geq 1 - \sum_{x \in \mathcal{S}'_\varepsilon} \Delta n_x + \sum_{x \in \mathcal{S}'_\varepsilon} x \cdot \Delta n_x \geq 1 - O(\varepsilon) + \sum_{x \in \mathcal{S}'_\varepsilon} x \cdot \Delta n_x.\end{aligned}$$

We proceed to bound $\sum_{x \in \mathcal{S}'_\varepsilon} x \cdot \Delta n_x$. Define $\Delta n_{\leq x}$ as $\sum_{x' \in \mathcal{S}'_\varepsilon : x' \leq x} \Delta n_{x'}$. Define $\Delta n_{<x}$ analogously. Note that $\Delta n_{\leq x}$ stays bounded between $[-\varepsilon, +\varepsilon]$. Let I denote the set of $x \in \mathcal{S}'_\varepsilon$ such that $\Delta n_{\leq x}$ changes sign, i.e., $\Delta n_{<x}$ and $\Delta n_{\leq x}$ have different signs. We assume w.l.o.g. that $\Delta n_{\leq x} = 0$ for any $x \in I$—we can do so by splitting Δn_x into two parts (and so, having two copies of x in $\mathcal{S}'_\varepsilon$). Observe that for any two consecutive $x_1, x_2 \in I$, the function $\Delta n_{\leq x}$ is unimodal as x varies from x_1 to x_2, i.e., it has only one local maxima or minima. This is because Δn_x is negative if $\Delta n_{<x}$ is positive, and *vice versa*.

Let the elements in I (sorted in ascending order) be $x_1, x_2, \ldots, x_q$. Let S_i denote the elements in $\mathcal{S}'_\varepsilon$ which lie between x_i and x_{i+1}, where we include x_{i+1} but exclude x_i. Note that $\sum_{x \in S_i} \Delta n_x = \Delta n_{\leq x_{i+1}} - \Delta n_{\leq x_i} = 0$. Let $x'_i = x_i + \varepsilon$ be the smallest element in S_i. Now observe that

$$\sum_{x \in S_i} x \cdot \Delta n_x = x'_i \cdot \sum_{x \in S_i} \Delta n_x + \sum_{x \in S_i} (x - x'_i) \cdot \Delta n_x = \sum_{x \in S_i} (x - x'_i) \cdot \Delta n_x.$$

Because of the unimodal property mentioned above, we get $\sum_{x \in S_i} |\Delta n_x| \leq 2\varepsilon$. Therefore, the absolute value of the above sum is at most $2\varepsilon \cdot (x_{i+1} - x'_i) \leq 2\varepsilon^2 |S_i|$, using that $x_{i+1} - x'_i = (|S_i| - 1)\varepsilon$. Now summing over all S_i we see that $\sum_x x \cdot \Delta n_x \leq O(\varepsilon)$ because $|\mathcal{S}_\varepsilon|$ is $O(1/\varepsilon)$. This proves the desired claim.

So to satisfy the first constraint we can increase n_0 by $O(\varepsilon)$. And then, increasing α'_ε by a further $O(\varepsilon)$ satisfies the remaining constraints, and proves the lemma. ■

Let $\tilde{n}_x$ be $i_x \cdot \varepsilon$ where i_x is an integer. Note that $\sum_x \tilde{n}_x \leq \alpha + O(\varepsilon) \leq 2$, so dividing through by ε, $\sum_x i_x \leq 2/\varepsilon$. Now for any index $x \in \{0\} \cup \mathcal{S}_\varepsilon$, we define $f_x := \frac{\tilde{n}_x}{\sum_{x'} \tilde{n}_{x'}} = \frac{i_x}{\sum_{x'} i_{x'}}$. If we set $T := \sum_x i_x \leq 2/\varepsilon$, then T is an integer at most $2/\varepsilon$, and f_x are integral multiplies of $1/T$, which satisfies the requirements of our algorithm. Next, we show that the dynamic solution maintained by our algorithm corresponds to a near-optimal solution to LP_ε.

Lemma 5.4.9 (Small Items Follow the LP) *Let $\varepsilon \leq 1/6$. Using $O(\varepsilon^{-2})$ worst-case recourse we can maintain packing of small items such that the content of all but $O(\varepsilon^{-2})$ designated bins in this packing form an $(\alpha^\star_\varepsilon + O(\varepsilon))$-feasible packing.*

Proof: The recourse bound is immediate, as each insertion or deletion causes a single item to move from at most $T \cdot 3/\varepsilon$ bins and $T = O(\varepsilon^{-1})$. For the rest of the argument, ignore the last bucket, contributing $O(\varepsilon^{-2})$ bins to our additive term. Let the total volume of items in the other bins be B. Since $\eta = \sum f_x b_x$ is the average bin-size, we expect to use $\approx B/\eta$ bins for these items. We now show that we use at most $(1 + O(\varepsilon)) \cdot \frac{f_x B}{\eta}$ and at least $(1 - O(\varepsilon)) \cdot \frac{f_x B}{\eta}$ bins of size b_x for each x.

Indeed, each (non-last) bucket satisfies the property that all bins in it, except perhaps for those in the last clump, are at least ε-close to the target value. Since each bucket has at least $1/\varepsilon$ clumps, it follows that if there are N clumps and the target average bin-size is η, then $(1 - \varepsilon)N$ clumps are at least $(\eta - \varepsilon)$ full on average. The total volume of a clump is $\eta \cdot T$, so $N \leq \frac{B}{(1-\varepsilon)(\eta-\varepsilon)\cdot T} = \frac{B}{\eta T}(1 + O(\varepsilon))$, where we use that $\eta \geq 1/4$. Therefore, the total number of bins of size b_x used is $f_x T \cdot N \leq (1 + O(\varepsilon)) \cdot \frac{B f_x}{\eta}$. The lower bound for the number of bins of size b_x follows from a similar argument and the observation that if we scale the volume of small items up by a factor of $(1 + O(\varepsilon))$, this volume would cause each bin to be filled to its target value. This implies that we use at least $(1 - O(\varepsilon)) \cdot \frac{B f_x}{\eta}$ bins with size b_x.

We now show that the $\bar{N}_x$ satisfy LPnew_ε with $\alpha^\star_\epsilon + O(\epsilon)$. Recall that we started with an optimal solution to LPnew_ε of value $\alpha^\star_\varepsilon + O(\varepsilon)$, used Lemma 5.4.8 to get $f_x = \tilde{n_x}/\sum_{x'} \tilde{n_{x'}}$ and ran the algorithm above. By the computations above, $\bar{N}_x$, the number of bins of size b_x used by our algorithm, is

$$(1 + O(\varepsilon)) \cdot \frac{f_x B}{\sum_{x'} f_{x'} b_{x'}} = (1 + O(\varepsilon)) \cdot \frac{\tilde{n}_x B}{\sum_{x'} \tilde{n}_{x'} b_{x'}} \leq (1 + O(\varepsilon)) \cdot \tilde{n}_x B,$$

where the last inequality follows from the fact that $\sum_{x'} \tilde{n}_{x'} b_{x'} \geq 1$ (by the first constraint of LPnew_ε). Likewise, by the same argument, we find that these $\bar{N}_x$ satisfy CR_ε with $\alpha_\epsilon = \alpha^\star_\varepsilon + O(\epsilon)$. Finally, since $\tilde{n}_x$ satisfies the constraints of LPnew_ε (up to additive $O(\varepsilon)$ changes in α_ε), we can verify that the quantities $\bar{N}_x$ satisfy the last two constraints of LP_ε (again up to additive $O(\varepsilon)$ changes in α_ε). To see that they also satisfy Vol_ε, we use the following calculation:

$$\sum_x \bar{N}_x \geq (1 - 3\varepsilon) \cdot \sum_x \frac{f_x B}{\sum_x b_x f_x} = (1 - O(\varepsilon)) \cdot \frac{B}{\sum_x b_x f_x} \geq (1 - O(\varepsilon)) \cdot B,$$

because $\sum_x f_x = 1$ and $b_x \leq 1$ for all x. Therefore, scaling all variables with $(1 - O(\varepsilon))$ will satisfy constraint Vol_ε as well. It follows that $\bar{N}_x$ satisfy LP_ε with $\alpha_\varepsilon = \alpha^\star_\varepsilon + O(\varepsilon)$. ■

Our Algorithm

Here we provide missing proofs of some of the claims made in our algorithm's analysis in §**??**.

Amortized Algorithm

In §**??** we claimed that if we start an epoch with a packing using $N \leq (\alpha + O(\varepsilon)) \cdot OPT(\mathcal{I}_t) + O(\varepsilon^{-2})$ bins, as we do (here $\mathcal{I}_t$ is the instance at the beginning of the epoch), then if for $\varepsilon \cdot N$ updates we address updates naïvely, our solution at any given time t' during the epoch uses at most $N_{t'} \leq (\alpha + O(\varepsilon)) \cdot OPT(\mathcal{I}_{t'}) + O(\varepsilon^{-2})$ bins. To see this, note that as each update can only change $OPT(\mathcal{I}_{t'})$ and the number of bins we use, $N_{t'}$, by one. Therefore, $N \cdot (1-\varepsilon) \leq N_{t'} \leq N \cdot (1+\varepsilon)$ and $OPT(\mathcal{I}'_t) \geq OPT(\mathcal{I}) + \varepsilon \cdot N$. By our upper bound on N, we obtain

$$\begin{aligned}
N_{t'} &\leq (1+\varepsilon) \cdot N \\
&\leq (\alpha + O(\varepsilon)) \cdot OPT(\mathcal{I}_t) + \varepsilon \cdot N + O(\varepsilon^{-2}) \\
&\leq (\alpha + O(\varepsilon)) \cdot (OPT(\mathcal{I}_{t'}) + \varepsilon \cdot N) + \varepsilon \cdot N + O(\varepsilon^{-2}) \\
&= (\alpha + O(\varepsilon)) \cdot OPT(\mathcal{I}_{t'}) + O(\varepsilon \cdot N) + O(\varepsilon^{-2}) \\
&\leq (\alpha + O(\varepsilon)) \cdot OPT(\mathcal{I}_{t'}) + O(\varepsilon \cdot N_{t'}) + O(\varepsilon^{-2}),
\end{aligned}$$

where we used in the last step the fact that $N \leq N_{t'}/(1-\varepsilon)$. Subtracting the $O(\varepsilon \cdot N_{t'})$ term from both sides of the above and dividing through by $(1 - O(\varepsilon))$, we obtain the claimed bound.

C.2 Omitted Proofs of Section 5.5 (General Movement Costs)

Here we provide proofs for our general amortized recourse upper and lower bounds, and discuss cases for which the algorithmic bounds can be made worst case (in §C.2.2).

C.2.1 Matching the Lower Bounds for Online Algorithms

Let $\beta \geq 2$. Any adversary process $\mathcal{B}$ showing a lower bound of c for the a.c.r of any online BIN PACKING algorithm can be converted into a fully-dynamic BIN PACKING instance with general movement costs such that any fully-dynamic BIN PACKING algorithm with amortized recourse at most β must have a.c.r at least c.

Proof:[Proof of Section 5.5.1] Take the adversary process $\mathcal{B}$, and use it to generate an instance for the fully-dynamic algorithm $\mathcal{A}$ as follows. (When there are k items in the system, let them be labeled $e_1, e_2, \ldots, e_k$.)

I: Given system in a state with k elements, use $\mathcal{B}$ to generate the next element e_{k+1}, having movement cost $(2\beta)^{-(k+1)}$.

II: If $\mathcal{A}$ places e_{k+1} into some bin and does not move any other item, go back to Step I to generate the next element.

III: However, if $\mathcal{A}$ moves some items, and e_j is the item with the smallest index that is repacked, delete items $e_{j+1}, \ldots, e_{k+1}$. This may in turn cause elements to be repacked, so delete all

items with indices strictly higher than than the smallest index item that is repacked. Eventually we stop at a state with $k' \geq 1$ elements such that only element $e_{k'}$ has been repacked. Now go back to Step I. (Also, $e_{k'+1}, \ldots, e_{k+1}$ are deemed undefined.)

Since the location of each item e_i is based only on the knowledge of prior elements in the sequence $e_1, e_2, \ldots, e_{i-1}$ and their bins, the resulting algorithm is another online algorithm. So if the length of the sequence eventually goes to infinity, we are guaranteed to reach an instance for which the a.c.r of this algorithm will be at least c. Hence we want to show that for any n, the length of the sequence eventually reaches n (or the adversary process stops, having showed a lower bound of c). Consider a potential function Φ which is zero when the system has no elements. When a new element is added by $\mathcal{B}$, we increase Φ by β times the movement cost for this element. Moreover, when $\mathcal{A}$ moves elements, we subtract the movement costs of these elements from Φ. Since $\mathcal{A}$ ensures an amortized recourse bound of β, the potential must remain non-negative.

For a contradiction, suppose the length of the sequence remains bounded by n. Hence, there is some length $k < n$ such that Step III causes the sequence to become of length k arbitrarily often. Note that the total increase in Φ between two such events is at most $\beta \sum_{i=k+1}^{n} (2\beta)^{-i} \leq \frac{(2\beta)^{-(k+1)}}{2\beta - 1}$. Since $\beta \geq 2$, this increase is strictly less than $(2\beta)^{-k}$, the total decrease in Φ due to the movement of element k alone. Since the potential decreases between two such events, there can only be finitely many such events, so the length of the sequence, i.e., the number of items in the system increases over time, eventually giving us the claimed lower bound. ■

C.2.2 (Nearly) Matching the Upper Bounds for Online Algorithms

We start by proving our lemma for packing similarly-sized items.

[Near-Uniform Sizes] There exists a fully-dynamic BIN PACKING algorithm with constant worst case recourse which given items of sizes $s_i \in [1/k, 1/(k-1))$ for some integer $k \geq 1$, packs them into bins of which all but one contain $k - 1$ items and are hence at least $1 - 1/k$ full. (If all items have size $1/k$, the algorithm packs k items in all bins but one.)

Proof: We round down movement costs of each item to the next-lower power of two. We maintain all items sorted by movement cost, with the costliest items in the first bin. All bins (except perhaps the last bin) contain $k - 1$ items; if all items have size $1/k$ then bins contain k items. Insertion and deletion of an item of cost $c_i = 2^\ell$ can be assumed to be performed at the last bin containing an item of this cost (possibly incurring an extra movement cost of c_i, by replacing item i with another item of cost c_i). If addition or deletion leaves a bin with one item too many or too few, we move a single item to/from the last bin containing an item of the next lower cost, and so on. Since items are sorted and the costs are powers of 2, the total movement cost is at most $c_i + c_i \cdot (1 + 1/2 + 1/4 + \ldots) = 3 \cdot c_i$; the loss due to rounding means the (worst case) recourse is ≤ 6. The lemma follows. ■

A Simple 2-approximate Solution

Suppose we round all item sizes to powers of 2 and use Lemma 5.5.2 on items of size at least $1/n$, where n is the maximum number of items in the instance $\mathcal{I}_t$ over all times t, while packing

all items of size less than $1/n$ into a single bin.[1] Then, we ensure that all but $1 + \log_2 n$ bins are full. This approach can be applied to general instances by rounding up item sizes to powers of two, yields the following fact.

Fact C.2.1 *There exists a fully-dynamic* BIN PACKING *algorithm with a.c.r* 2 *and additive term* $1 + \log_2 n$*, using constant worst case recourse.*

However, this simple result is highly unsatisfactory for two reasons. Firstly, as we shall show, its a.c.r is suboptimal. The second reason concerns its additive term, which can be blown up to be arbitrarily large without effecting the optimal solution in any significant way, by adding N items of size $1/N$ for arbitrarily large N. In what follows, we will aim to design algorithms with better a.c.r and both additive term and recourse independent of n.

The Super Harmonic family of online algorithms

A *Super-harmonic* (abbreviated as SH) algorithm consists of a partition of the unit interval $[0, 1]$ into $K + 1$ intervals, $[0, \varepsilon], (t_0 = \varepsilon, t_1](t_1, t_2], \ldots, (t_{K-1}, t_K = 1]$. Small items (i.e., items of size at most ε) are packed using FIRSTFIT into dedicated bins; because the items are small, all but one of these are at least $1 - \varepsilon$ full. For larger items, each arriving item is of type i if its size is in the range $(t_{i-1}, t_i]$. Items are colored either blue or red by the algorithm, with each bin containing items of at most two distinct item types i and j. If a bin contains only one item type, its items are all colored the same, and if a bin contains two item types i and j, then all items of type i are colored blue and items of type j are colored red (or vice versa). The SH algorithm has associated with it three number sequences $(\alpha_i)_{i=1}^K, (\beta_i)_{i=1}^K, (\gamma_i)_{i=1}^K$, and an underlying bipartite *compatibility graph* $\mathcal{G} = (V, E)$ whose role will be made clear shortly. A bin with blue (resp., red) type i items contains at most β_i (resp., γ_i) items of type i, and is *open* if it contains less than β_i type i (resp., less than γ_j type j items). The compatibility graph determines which pair of (colored) item types can share a bin. The compatibility graph $\mathcal{G} = (V, E)$ is defined on the vertex set $V = \{b_i \mid i \in [K]\} \cup \{r_i \mid j \in [K]\}$, with an edge $(b_i, r_j) \in E$ indicating blue items of type i and red items of type j are *compatible*; they are allowed to be placed in a common bin. Apart from the above properties, an SH algorithm must satisfy the following invariants. We restate the invariants:

(P1) 1 The number of open bins is $O(1)$.

(P2) 2 If n_i is the number of type-i items, the number of red type-i items is $\lfloor \alpha_i \cdot n_i \rfloor$.

(P3) 3 If $(b_i, r_j) \in E$ (blue type i items and red type j items are compatible), there is no pair of bins with one containing nothing but blue type i items and one containing nothing but red type j items.

The above invariants allow one to bound the asymptotic competitive ratio of an SH algorithm, depending on the choices of $(\alpha_i)_{i=1}^K, (\beta_i)_{i=1}^K, (\gamma_i)_{i=1}^K$ and the compatibility graph $\mathcal{G}$. In particular, Seiden [131] showed the following.

Lemma C.2.2 (Seiden [131]) *There exists an SH algorithm with a.c.r* 1.58889.

[1]We assume we know n. If n is unknown, we can obtain the same amortized recourse by a simple "guess and double/halve" approach.

A particular property the SH algorithm implied by Lemma C.2.2 which we will make use of later is that its parameters or inverses are all at most a constant; in particular, $\varepsilon^{-1} = O(1)$ and $K = O(1)$, and similarly $\beta_i, \gamma_i = O(1)$ for all $i \in [K]$.

In the following sections we proceed to describe how to maintain the above invariants that suffice to bound the competitive ratio of an SH algorithm. We start by addressing the problem of maintaining the SH invariants for the large items, in Section C.2.2. We then show how to pack small items (i.e., items of size at most ε) into bins such that all but a constant of these bins are at least $1 - \varepsilon$ full, in Section C.2.2.[2] Finally we conclude with our upper bound in Section C.2.2.

SH Algorithms: Dealing with Large Items

First, we round all movement costs to powers of 2, increasing our recourse cost by at most a factor of 2. Now, our algorithm will have recourse cost which will be some function of $K, (\beta_i)_{i=1}^K, (\gamma_i)_{i=1}^K$; as for our usage, we have $K = O(1)$, and $\beta_i, \gamma_i = O(1)$ for all $i \in [K]$, we will simplify notation and assume that these values are indeed all bounded from above by a constant. Consequently, when moving around groups of up to β_i blue (resp. γ_i red) type i items whose highest movement cost is c, the overall movement cost will be $O(c)$. The stipulation that $K = O(1)$ will prove useful shortly. We now explain how we maintain the invariants of SH algorithms.

Satisfying Property 1. We keep all blue items (resp. red items) of type i in bins containing up to β_i items (resp. γ_i items). We sort all bins containing type-i items by the cost of the costliest type-i item in the bin, where only the last bin containing type-i items contains less than β_i blue (alternatively, γ_i red) type-i items. Therefore, if we succeed in maintaining the above, the number of open bins is $O(1)$; i.e., Property 1 is satisfied. We explain below how to maintain this property together with Properties 2 and 3.

Satisfying Property 2. We will only satisfy Property 2 *approximately*, such that the number of red type-i items is in the range $[\lfloor \alpha_i \cdot n_i \rfloor, \lfloor \alpha_i \cdot n_i \rfloor + \delta_i]$, where $\delta_i = \max\{\alpha_i(\beta_i - \gamma_i) + 1 + \gamma_i, 0\}$. Removing these at most δ_i red type-i items results in a smaller bin packing instance, and a solution requiring the same number of bins, satisfying the invariants of SH algorithms. Notice that this change to the solution can change the number of open bins by at most $\sum_i \delta_i \leq \sum_i \alpha_i \beta_i + K = O(1)$. Therefore, satisfying Property 2 approximately suffices to obtain our sought-after a.c.r. In fact, we satisfy a stronger property: each prefix of bins containing type-i items has a number of red type-i items in the range $[\lfloor \alpha_i \cdot n'_i \rfloor, \lfloor \alpha_i \cdot n'_i \rfloor + \delta_i]$, where n'_i is the number of type-i items in the prefix.

Maintaining the above prefix invariant on deletion is simple enough: when removing some type-i item of cost 2^k, we move the last type-i item of the next movement cost to this item's place in the packing, continuing until we reach a type-i item with no cheaper type-i items. The movement cost here is at most $2^k + 2^{k-1} + 2^{k-2} + \cdots = O(2^k)$, so the (worst case) recourse is constant. As

[2] Strictly speaking, we will only pack small items into bins which are $1 - \epsilon$ full *on average*. However, redistributing these small items or portions of these items within these bins will make these bins $1 - \epsilon$ full without changing the number of bins used by the solution. The bounds on SH algorithms therefore carries through.

the prefix invariant was satisfied before deletion, it is also satisfied after deletion, as we effectively only remove an item from the last bin. When inserting a type-i item, we insert the item into the last appropriate bin according to the item's movement cost. This might cause this bin to overflow, in which case we take the cheapest item in this bin and move it into the last appropriate bin according to this item's cost, continuing in this fashion until we reach an open bin, or are forced to open a new bin. (The recourse here is a constant, too). The choice of color for type-i items in a newly-opened bin depends on the number of type-i items before this insertion, n_i, and the number of red type-i items before this insertion, m_i. If $m_i + \gamma_i \leq \lfloor \alpha_i(n_i + \gamma_i) \rfloor + \delta_i$, the new bin's red items are colored red. By this condition, the number of red items for the following γ_i insertions into this bin will satisfy our prefix property. If the condition is not satisfied, the bin is colored blue. Now, as

$$\begin{aligned} m_i + \gamma_i &> \lfloor \alpha_i(n_i + \gamma_i) \rfloor + \delta_i \\ &\geq \lfloor \alpha_i(n_i + \beta_i) \rfloor + \alpha_i(\gamma_i - \beta_i) - 1 + \delta_i \\ &\geq \lfloor \alpha_i(n_i + \beta_i) \rfloor - 1 + \gamma_i \end{aligned}$$

we find that after this new bin contains β_i type i items, the number of red items in the prefix is m_i while the number of type-i items is $n_i' = n_i + \beta_i$, and so this prefix too satisfies the prefix condition.

We conclude that the above methods approximately maintain Property 2 (as well as Property 1) while only incurring constant worst case recourse.

Satisfying Property 3. Finally, in order to satisfy Property 3, we consider the groups of up to β_i and γ_j blue and red items of type i and j packed in the same bin as nodes in a bipartite graph. A blue type i (resp. red type j) group is a copy of node b_i (resp. r_j) in the compatibility graph $\mathcal{G}$, and copies of nodes b_i and r_j are connected if they are connected in $\mathcal{G}$. If a particular node b_i and r_j are placed in the same bin, then we treat that edge as matched. Each node has a cost which is simply the maximum movement cost of an item in the group which the node represents. All nodes have a preference order over their neighbors, preferring a costlier neighbor, while breaking ties consistently. We will maintain a stable matching in this subgraph, where two nodes are matched if the items they represent. This stability clearly implies Property 3. We now proceed to describe how to maintain this bipartite graph along with a stable matching in it.

The underlying operations we will have is addition and removal of a node of red or blue type i items of cost 2^k; i.e., with costliest item having movement cost 2^k. As argued before, as each group contains $O(1)$ items, the movement cost of moving items of such a group is $O(2^k)$. Using this operation we can implement insertion and deletion of single items, by changing the movement cost of groups with items moved, implemented by removal of a group and re-insertion with a higher/lower cost if the cost changes (or simple insertion/removal for a new bin opened/bin closed). As argued above, the cost of items moved is during updates in order to satisfy Property 2 is $O(2^k)$, where 2^k is the cost of the item added/removed; consequently, the costs of removals and insertions of groups is $O(2^k)$. We therefore need to show that the cost of insertion/removal of a group of cost 2^k is $O(2^k)$.

Insertions and deletions of blue and red nodes is symmetric, so we consider insertions and deletions of a blue node b only. Upon insertion of some node b of cost 2^k, we insert b into its place in the ordering, and scan its neighbors for the first neighbor r which strictly prefers b to its current

match, b'. If no such r exists, we are done. If such an r exists, b' is unmatched from r (its items are removed from its bin) and b is matched to r (the items of b are placed in the same bin as r's bin). As b' has strictly less than b, its cost is at most 2^{k-1}. We now proceed similarly for b' as though we inserted b' into the graph. The overall movement cost is at most $2^k + 2^{k-1} + 2^{k-2} + \cdots = O(2^k)$.

Upon deletion of a blue node b of cost 2^k, if it had no previous match, we are done. If b did have a match r, this match scans its neighbors, starting at b, for its first neighbor b' of cost at most 2^k which prefers r to its current match. If the cost of b' is 2^k, we match r to the last blue node of the same type as b', denoted by b''. If b'' was previously matched, we proceed to match its match as if b'' were removed (i.e., as above). As every movement decreases the number of types of blue nodes of a given cost to consider, the overall movement cost is at most $K \cdot (2^k + 2^{k-1} + 2^{k-2} + \dots) = O(K \cdot 2^k) = O(2^k)$, where here we rely the number of types being $K = O(1)$.

We conclude that Property 3 can be maintained using constant worst case recourse.

Lemma C.2.3 *Properties 1, 2 and 3 can be maintained using constant worst-case recourse.*

SH Algorithms: Dealing with Small Items

Here we address the problem of packing small items into bins so that all but a constant number of bins are kept at least $1 - \varepsilon$ full. Lemma 5.5.2 allows us to do just this for items of size in the range $[\varepsilon', \varepsilon]$, for $\varepsilon' = \Omega(\varepsilon)$. Specifically, considering all integer values c in the range $[\lceil \frac{1}{\varepsilon} \rceil, \lceil \frac{1}{\varepsilon'} \rceil]$, then, as $\varepsilon^{-1} = O(1)$ and consequently $\lceil \frac{1}{\varepsilon'} \rceil - \lceil \frac{1}{\varepsilon} \rceil = O(1)$, we obtain the following.

Corollary C.2.4 *All items in the range $[\varepsilon', \varepsilon]$ for any $\varepsilon' = \Omega(\varepsilon)$ can be packed into bins which are all (barring perhaps $O(\epsilon^{-1}) = O(1)$ bins) at least $1 - \varepsilon$ full.*

It now remains to address the problem of efficiently maintaining a packing of items of size at most ε' into bins which are at least $1 - \varepsilon$ full (again, up to some $O(1)$ possible additive term). As $\varepsilon' = \Omega(\varepsilon)$, we will attempt to pack these items into bins which are at least $1 - O(\varepsilon')$ full on average. For notational simplicity from here on, we will abuse notation and denote ε' by ε, contending ourselves with a packing which is $1 + O(\varepsilon)$-competitive, and is therefore keeps bins $1 - O(\varepsilon)$ full on average (as OPT is close to the volume bound for instance made of only small items).

Let us first give the high-level idea of the algorithm before presenting the formal details. Define the *density* of an item as c_j/s_j, the ratio of its movement cost to its size. We arrange the bins in some fixed order, and the items in each bin will also be arranged in the order of decreasing density. This means the total order on the items (consider items in the order dictated by the ordering of bins, and then by the ordering within each bin) is in decreasing order of density as well. Besides this, we want all bins, except perhaps the last bin, to be approximately full (say, at least $1 - O(\varepsilon)$ full). The latter property will trivially guarantee $(1 + O(\varepsilon))$-competitive ratio. When we insert an item j, we place it in the correct bin according to its density. If this bin overflows (i.e., items in it have total size more than 1), then we remove some items from this bin (the ones with least density) and transfer them to the next bin – these items will have only smaller density than j, and so, their movement cost will be comparable to that of j. If the next bin can accommodate these items, then we can stop the process, otherwise this could lead to a long cascade. To prevent such long cascades, we arrange the bins in *buckets* – each bucket consists of about $O(\varepsilon^{-1})$ consecutive

bins, and all these buckets are approximately full except for the last bin in the bucket. Again, it is easy to see that this property will ensure $(1 + O(\varepsilon))$-competitive ratio. Note that this extends the idea of Berndt et al. Once we have these buckets, the above-mentioned cascade stops when we reach the last bin of a bucket. Consequently we have cascades of length at most $O(\varepsilon^{-1})$. If the last bin also overflows, we will add another bin at the end of this bucket, and if the bucket now gets too many bins, we will split it into two smaller buckets. One proceeds similarly for the case of deletes – if an item is deleted, we *borrow* some items from the next bin in the bucket, and again this could cascade only until the last bin in the bucket. (If the bucket ever has too few bins, we merge it with the next bucket).

However, this cascade is not the only issue. Because items are atomic and have varying sizes, it is possible that insertion of a tiny item (say of size $O(\varepsilon^2)$) could lead us to move items of size $O(\varepsilon)$. In this case, even though the density of the latter item is smaller than the inserted item, its total movement cost could be much higher. To prevent this, we ensure that whenever a bin overflows, we move out enough items from it so that it has $\Omega(\varepsilon)$ empty space. Now, the above situation will not happen unless we see tiny items amounting to a total size of $\Omega(\varepsilon)$. In such a case, we can charge the movement cost of the larger item to the movement cost of all such tiny items.

The situation with item deletes is similar. When a tiny item is deleted, it is possible that the corresponding bin underflows, and the item borrowed from the next bin is large (i.e., has size about ε). Again, we cannot bound the movement cost of this large item in terms of that of the item being deleted. To take care of such issues, we do not immediately remove such tiny items from the bin. We call such items *ghost* items – they have been deleted, but we have not removed them from the bins containing them. When a bin accumulates ghost items of total size about $\Omega(\varepsilon)$, we can afford to remove all these from the bin, and the total movement cost of such items can pay for borrowing items (whose total size would be $O(\varepsilon)$) from the next bin.

The analysis of the movement cost is done via by a potential function argument, to show the following result (whose proof appears in Section C.2.2): For all $\varepsilon \leq \frac{1}{6}$ there exists an asymptotically $(1 + O(\varepsilon))$-competitive bin packing algorithm with $O(\varepsilon^{-2})$ amortized recourse if all items have size at most ε.

We first describe the algorithm formally. Let B_i denote the items stored in a bin i. As mentioned above, our algorithm maintains a solution in which items are stored in decreasing order of density. I.e., for all $i < i'$, for every pair of jobs $j \in B_i$ and $j' \in B_{i'}$ we will have $c_j/v_j \geq c_{j'}/v_{j'}$. Recall that B_i could contain ghost jobs. Let A_i denote the jobs in B_i which have not been deleted yet (i.e., are not ghost jobs), and G_i denote the ghost jobs. We shall use $s(B_i)$ to denote the total size of items in B_i (define $s(A_i)$ similarly). We maintain the following invariants, satisfied by all bins B_i that are not the last bin in their bucket:

$$\begin{aligned} &\mathrm{P}_0 : 1 - 3\varepsilon \leq s(B_i) \leq 1. \\ &\mathrm{P}_1 : s(A_i) \geq 1 - 4\varepsilon. \end{aligned} \tag{C.3}$$

Finally, a bin B_i which is the last bin in its bucket has no ghost jobs. That is, $s(G_i) = 0$. Each bucket has at most $3/\varepsilon$ bins. Furthermore, each bucket, except perhaps for the last bucket, has at least ε^{-1} bins.

Our algorithm is given below. We use two functions GROWBUCKET(U) and SPLITBUCKET(U)

in these procedures, where U is a bucket. The first function is called when the bucket U *underflows*, i.e., when U has less than ε^{-1} bins. If U is the last bucket, then we need not do anything. Otherwise, let U' be the bucket following U. We merge U and U' into one bucket (note that the last bin of U need not satisfy the invariant conditions above, and so, we will need to do additional processing to ensure that the conditions are satisfied for this bin). The function SPLITBUCKET(U) is called when U contains more than $3/\varepsilon$ bins. In this case, we split it into two buckets, each of size more than ε^{-1}.

Algorithm INSERT(j)

Add job j into appropriate bin i
if $s(B_i) > 1$ **then**
    ERASEGHOST(i, s_j)
if $s(B_i) > 1$ **then**
    OVERFLOW(i, $s(B_i) - 1 + 2\varepsilon$)

Algorithm DELETE(j)

Let i be the bin containing j
if i is the last bin in its bucket **then**
    Erase j from i
else
    Mark j as a ghost job.
    if $s(A_i) < 1 - 4\varepsilon$ **then**
        Erase all ghost jobs from bin i
        BORROW(i, $1 - 3\varepsilon - s(A_i)$)

Algorithm OVERFLOW(i, v)

Let X be the minimum density jobs in
    B_i s.t. $s(X) \geq v$
if i is the last bin its bucket or
    $v \geq 1 - 3\varepsilon$ **then**
    Add a new bin i' after i in this bucket
    Move X from i to i'.
    Let U be the bucket containing i.
    if U has more than $3/\varepsilon$ bins **then**
        SPLITBUCKET(U)
else
    Move X from bin i to bin $i + 1$
    if $s(B_{i+1}) > 1 - \varepsilon$ **then**
        ERASEGHOST($i + 1, s(B_{i+1}) - 1 + 2\varepsilon$)
    if $s(B_{i+1}) > 1 - \varepsilon$ **then**
        OVERFLOW($i + 1, s(B_{i+1}) - 1 + 2\varepsilon$)

Algorithm BORROW(i, v)

Let X be the minimum density jobs in B_{i+1}
    s.t. $s(X \cap A_{i+1}) \geq \min(v, s(A_{i+1})$
Remove X from B_{i+1} (erase ghosts in X)
Move $A \triangleq X \cap A_{i+1}$ to B_i
if bin $i + 1$ is empty **then**
    Remove this bin from its bucket U
    if U has less than ε^{-1} bins and
        it is not the last bucket **then**
        GROWBUCKET(U)
        if U has more than $3/\varepsilon$ bins **then**
            SPLITBUCKET(U)
    if $s(A_i) < 1 - 3\varepsilon$ **then**
        BORROW(i, $1 - s(A_i) - 3\varepsilon$)
else
    if $s(A_{i+1}) < 1 - 3\varepsilon$ **then**
        Erase all ghost jobs from bin $i + 1$
        BORROW($i + 1$, $1 - 3\varepsilon - s(A_{i+1})$)

We first describe the function INSERT(j) for an item j. We insert job j into the appropriate bin i (recall that the items are ordered by their densities). If this bin overflows, then we call the procedure ERASEGHOST(i, s_j). The procedure ERASEGHOST(i, s), where i is a bin and s is a

positive quantity, starts erasing ghost jobs from B_i till one of the following events happen: (i) B_i has no ghost jobs, or (ii) total size of ghost jobs removed exceeds s. Since all jobs are of size at most ε, this implies that the total size of ghost jobs removed is at most $\min(s(G_i), s + \varepsilon)$. Now its possible that even after removing these jobs, the bin overflows (this will happen only if $s(G_i)$ was at most s_j). In this case, we offload some of the items (of lowest density) to the next bin in the bucket. Recall that when we do this, we would like to create $O(\varepsilon)$ empty space in bin i. So we transfer jobs of least density in B_i of total size at least $s(B_i) - 1 + 2\varepsilon$ to the next bin (since all jobs are small, the empty space in bin i will be in the range $[2\varepsilon, 3\varepsilon]$) . This is done by calling the procedure OVERFLOW(i, $s(B_i) - 1 + 2\varepsilon$). The procedure OVERFLOW(i, v), where i is a bin and v is a positive quantity, first builds a set X of items as follows – consider items in B_i in increasing order of density, and keep adding them to X till the total size of X, denoted by $s(X)$, exceeds v (so, the total size of X is at most $v + \varepsilon$). We now transfer X to the next bin in the bucket (note that by construction, X does not have any ghost jobs). The same process repeats at this bin (although we will say that overflow occurs at this bin if $s(B_i)$ exceeds $1 - \varepsilon$). This cascade can end in several ways – it is not difficult to show that between two consecutive calls to OVERFLOW, the parameter v grows by at most ε. If v becomes larger than $1 - 3\varepsilon$, we just create a new bin and assign all of X to this new bin. If we reach the last bin, we again create a new bin and add X to it. In both these cases, the size of the bucket increases by 1, and so we may need to split this bucket. Finally, it is possible that even after transfer of X, $s(A_i)$ does not exceed $1 - \varepsilon$ – we can stop the cascade at this point.

Next we consider the case of deletion of an item j. Let i be the bin containing j. If i is the last bin in its bucket, then we simply remove j (recall that the last bin in a bucket cannot contain ghost jobs). Otherwise, we mark j as a ghost job. This does not change $s(B_i)$, but could decrease $s(A_i)$. If it violates property P_1, we borrow enough items from the next bin such that i has free space of about 2ε only. The function BORROW(i, v), where i is a bin and v is a positive quantity, borrows densest (non-ghost) items from the next bin $i + 1$ of total size at least v. So it orders the (non-ghost) items in $i + 1$ in decreasing density, and picks them till the total size accumulated is at least v. This process may cascade (and will stop before we reach the last bin). However there is one subtlety – between two consecutive calls to BORROW, the value of the parameter v may grow (by up to ε), and so, it is possible that bin $i + 1$ becomes empty, and we are not able to transfer enough items from $i + 1$ to i. In this case, we first remove $i + 1$, and continue the process (if the size of the bucket becomes too small, we handle this case by merging it with the next bin and splitting the resulting bucket if needed). Since we did not move enough items to i, we may need to call BORROW(i, v') again with suitable value of v'. There is a worry that the function BORROW(i, v) may call BORROW(i, v') with the same bin i, and so, whether this will terminate. But note that, whenever this case happens, we delete one bin, and so, this process will eventually terminate.

Analysis

We begin by showing properties of the OVERFLOW and the BORROW functions.

Lemma C.2.5 *Whenever* OVERFLOW*(i, v) is called, bin i has no ghost jobs. Furthermore, $s(B_i) = v + 1 - 2\varepsilon$. Finally, when* OVERFLOW*(i, v) ends, $1 - 3\varepsilon \leq s(B_i) \leq 1 - 2\varepsilon$, and $s(G_i) = 0$.*

Similarly, whenever BORROW*(i, v) is called, bin i has no ghost jobs. Furthermore, $s(B_i) =$*

$v + 1 - 3\varepsilon$. *Finally, when* BORROW*(i, v) ends, either i is the last bin in its bucket or* $1 - 3\varepsilon \leq s(B_i) \leq 1 - 2\varepsilon$*, and* $s(G_i) = 0$.

Proof: When we insert an item j in a bin i, we erase ghost items from i till either (i) we erase all ghost items, or (ii) we erase ghost items of total size at least s_j. If the second case happens, then the fact that $s(B_i) \leq 1$ before insertion of item j implies that we will not call OVERFLOW in Line 3 of INSERT(j). Therefore, if we do call OVERFLOW, it must be the case that we ended up deleting all ghost jobs in i. Further we call OVERFLOW with $v = s(B_i) - 1 + 2\varepsilon$. Similarly, before we call OVERFLOW$(i+1, v)$ in Line 15, we try to ensure ghost jobs of the same volume v from bin $(i+1)$. If we indeed manage to remove ghost jobs of total size at least v, we will not make a recursive call to OVERFLOW. This proves the first part of the lemma.

When OVERFLOW(i, v), terminates, we make sure that we have transferred the densest v volume out of i to the next bin. Since job sizes are at most ε, we will transfer at most items of total size in the range $[v, v+\varepsilon]$ out of i. Since $s(B_i) = v+1-2\varepsilon$ before calling this function, it follows that $s(B_i)$ after end of this function lies in the range $[1 - 3\varepsilon, 1 - 2\varepsilon]$. The claim for the BORROW borrow function follows similarly. ■

The following corollary follows immediately from the above lemma.

Corollary C.2.6 *Properties* P_0 *and* P_1 *from (C.3) are satisfied throughout by all bins which are not the last bins in their bucket. All bins B_i which are last in their bucket satisfy $s(G_i) = 0$.*

We are now ready to prove the claimed bounds obtained by our algorithm for small items.

Proof:[Proof of Lemma 5.5.2] First we bound the competitive ratio. Barring the last bucket, which has $O(\varepsilon^{-1})$ bins, all other buckets have at least ε^{-1} bins. All bins (except perhaps for the last bin) in each of these buckets have at most 4ε space that is empty or is filled by ghost jobs. Therefore, the competitive ratio is $(1 + O(\varepsilon))$ with an additive $O(\varepsilon^{-1})$ bins.

It remains to bound the recourse cost of the algorithm. For a solution $\mathcal{S}$, define the potential:

$$\Phi(\mathcal{S}) = \frac{4}{\varepsilon^2} \sum_{\text{bins } i} \Phi(i), \quad \text{where}$$

$$\Phi(i) = c(G_i) + c(\text{fractional sparsest items above } 1 - \varepsilon \text{ in } B_i).$$

The second term in the definition of $\Phi(i)$ is evaluated as follows: arrange the items in B_i in decreasing order of density, and consider the items occupying the last ε space in bin i (the total size of these items could be less than ε if the bin is not completely full). Observe that at most one item may be counted fractionally here. The second term is simply the sum of the movement costs of all such item, where a fractional item contributes the appropriate fraction of its movement cost.

Now we bound the amortized movement cost with respect to potential Φ. First consider the case when we insert item j in bin i. This could raise Φ_i by c_j. If bin i does not overflow, we do not pay any movement cost. Further, deletion of ghost jobs from bin i can only decrease the potential. Therefore, the amortized movement cost is bounded by c_j. On the other hand, suppose bin i overflows and this results in calling the function OVERFLOW(i, v) with a suitable v. Before this function call, let I denote the set of items which are among the sparsest items above $(1 - \varepsilon)$ volume in bin i (i.e., these contribute towards Φ_i). Let d be the density of the least density item in I. Since $s(B_i) \geq 1$, it follows that $\Phi_i \geq d\varepsilon$. When this procedure ends, Lemma C.2.5 shows that $s(B_i) \leq 1 - \varepsilon$, and so, Φ_i would be 0. Thus, Φ_i decreases by at least $d\varepsilon - c_j$. Lemma C.2.5 also

shows that if we had recursively called OVERFLOW(i', v') for any other bin i', then $s(B_{i'})$ would be at most $1 - \varepsilon$, and so, $\Phi_{i'}$ would be 0 when this process ends. It follows that the overall potential function Φ decreases by at least $4(d\varepsilon - c_j)/\varepsilon^2$. Let us now estimate the total movement cost. We transfer items of total size at most 1 from one bin to another. This process will clearly end when we reach the end of the bucket, and so the total size of items moved during this process is at most $3/\varepsilon$, i.e., the number of bins in this bucket. The density of these items is at most d, and so the total movement cost is at most $3d/\varepsilon$. Thus, the amortized movement cost is at most c_j/ε^2.

Now we consider deletion of an item j which is stored in bin i. Again the interesting case is when this leads to calling the function BORROW. Before j was deleted, $s(B_i)$ was at least $1 - 3\varepsilon$ (property P_0). After we mark j as a ghost job, $s(A_i)$ drops below $1 - 4\varepsilon$. So the total size of ghost jobs is at least ε. Since we are removing all these jobs from bin i, Φ_i decreases by at least $d\varepsilon$, where d is the density of the least density ghost job in i. As in the case of insert, Lemma C.2.5 shows that whenever we make a function call BORROW(i', v'), $\Phi_{i'}$ becomes 0 when this process ends. So, the potential Φ decreases by at least $4d/\varepsilon$. Now, we count the total movement cost. We transfer items of size at most 1 between two bins, and so, we just need to count how many bins are affected during this process (note that if we make several calls to BORROW with the same bin i', the the *total* size of items transferred to i' is at most 1). Let U be the bucket containing i. If we do not call SPLITBUCKET(U), then we affect at most $3/\varepsilon$ bins. If we call SPLITBUCKET(U), then it must be the case that U had only ε^{-1} bins. When we merge U with the next bucket, and perhaps split this merged bucket, the new bucket U has at least $2/\varepsilon$ bins, and so, we will not call SPLITBUCKET(U) again. Thus, we will touch at most $3/\varepsilon$ buckets in any case. It follows that the total movement cost is at most $3d/\varepsilon$ (all bins following i store items of density at most d). Therefore, the amortized movement cost is negative. This proves the desired result. ■

Dealing with Small Items: Summary. Combining Corollary C.2.4 and Lemma 5.5.2, we find that we can pack small items into bins which, ignoring some $O(1)$ many bins are $1 - \varepsilon$ full on average. Formally, we have the following.

Lemma C.2.7 *For all $\varepsilon \leq \frac{1}{6}$ there exists a fully-dynamic bin packing algorithm with $O(\frac{1}{\varepsilon^2})$ amortized recourse for instances where all items have size at most ε which packs items into bins which, ignoring some $O(1)$ bins, are at least $1 - \varepsilon$ full on average.*

Worst case bounds. Note that the above algorithm yields worst case bounds for several natural scenarios, given in the following corollaries.

Corollary C.2.8 *For all $\varepsilon \leq \frac{1}{6}$ there exists a fully-dynamic bin packing algorithm with $O(\frac{1}{\delta \cdot \varepsilon^2})$ worst case recourse for instances where all items have size in the range $[\delta, \varepsilon]$ which packs items into bins which, ignoring some $O(1)$ bins, are at least $1 - \varepsilon$ full on average.*

Proof:[Proof (Sketch)] The algorithm is precisely the algorithm of Lemma C.2.7, only now in our analysis, as each item has size δ, any single removal can incur movement of at most ϵ^2 size, by our algorithm's definition. The worst case migration factor follows. ■

Finally, if we group the items of size less than $(1/n, \epsilon]$ into ranges of size $(2^i, 2i + 1]$ (guessing and doubling n as necessary) requires only $O(\log n)$ additive bins (one per size range), while allowing worst case recourse bounds by the previous corollary.

Corollary C.2.9 *For all $\varepsilon \leq \frac{1}{6}$ there exists a fully-dynamic bin packing algorithm with $O(\frac{1}{\varepsilon^2})$ worst case recourse for instances where all items have size at most ε which packs items into bins which, ignoring some $O(\log n)$ bins, are at least $1-\varepsilon$ full on average.*

SH Algorithms: Conclusion

§C.2.2 and §C.2.2 show how to maintain all invariants of SH algorithms, using $O(1)$ amortized recourse, provided $\varepsilon^{-1} = O(1)$ and $K = O(1)$, and $\beta_i, \gamma_i = O(1)$ for all $i \in [K]$. As the SH algorithm implied by Lemma C.2.2 satisfies these conditions, we obtain this section's main positive result.

There exists a fully-dynamic BIN PACKING algorithm with a.c.r 1.58889 and constant additive term using constant recourse under general movement costs.

C.3 Omitted Proofs of Section 5.6 (Size Movement Costs)

Here we provide proofs for our matching amortized size cost upper and lower bounds.

C.3.1 Amortized Migration Factor Upper Bound

We start with a description and analysis of the trivial optimal algorithm for size costs.

Fact 5.6.1 *For all $\epsilon \leq 1/2$, there exists an algorithm requiring $(1+O(\varepsilon)) \cdot OPT(\mathcal{I}_t) + O(\varepsilon^{-2})$ bins at all times t while using only $O(\varepsilon^{-1})$ amortized migration factor.*

Proof: We divide the input into *epochs*. The first epoch starts at time 0. For an epoch starting at time t, let V_t be the total volume of items present at time t. The epoch starting at time t ends when the total volume of items inserted or deleted during this epoch exceeds εV_t. We now explain the bin packing algorithm. Whenever an epoch ends (say at time t), we use an offline A(F)PTAS (e.g., [46] or [101]) to efficiently compute a solution using at most $(1+\varepsilon) \cdot OPT(\mathcal{I}_t) + O(\varepsilon^{-2})$ bins. (Recall that $\mathcal{I}_t$ denotes the input at time t.) We pack items arriving during an epoch in new bins, using the first-fit algorithm to pack them. If an item gets deleted during an epoch, we *pretend* that it is still in the system and continue to pack it. When an epoch ends, we remove all the items which were deleted during this epoch, and recompute a solution using an off-line A(F)PTAS algorithm as indicated above.

To bound the recourse cost, observe that if the starting volume of items in an epoch starting at time t is V_t, the volume at the end of this epoch is at most $V_t + A_t$, where A_t is the volume of items that arrived and departed during this epoch. As $A_t > \varepsilon V_t$, the cost of reassigning these items of volume at most $V_t + A_t$ can be charged to A_t. Specifically, the amortized migration cost of the epoch starting at time t is at most $(V_t + A_t)/A_t < V_t/\epsilon V_t + 1 = O(\varepsilon^{-1})$. Consequently, the overall amortized migration cost is $O(\varepsilon^{-1})$.

To bound the competitive ratio, we use the following easy fact: the optimal number of bins to pack a bin packing instance of total volume V lies between V and $2V$. (E.g., the first-fit algorithm achieves this upper bound.) Consider an epoch starting at time t. Let V_t denote the volume of the input $\mathcal{I}_t$ at time t. The algorithm uses at most $(1+\varepsilon) \cdot OPT(\mathcal{I}_t) + O(\varepsilon^{-2})$ bins at time t.

Consider an arbitrary time t' during this epoch. As a packing of the instance $\mathcal{I}_{t'}$ can be extended to a packing of $\mathcal{I}_t$ by packing $\mathcal{I}_t \setminus \mathcal{I}_{t'}$ with the first fit algorithm, using a further $2\varepsilon V_t$ bins, we have $OPT(\mathcal{I}_t) \leq OPT(\mathcal{I}_{t'}) + 2\varepsilon V_t$. On the other hand, our algorithm uses at most an additional $2\varepsilon V_t$ bins at time t' compared to the beginning of the epoch. Therefore, the number of bins used at time t' is at most

$$\begin{aligned}(1+\varepsilon)\cdot OPT(\mathcal{I}_t) + 2\varepsilon V_t + O(\varepsilon^{-2}) &\leq (1+\varepsilon)\cdot(OPT(\mathcal{I}_{t'}) + 2\varepsilon V_t) + 2\varepsilon V_t + O(\varepsilon^{-2}) \\ &\leq (1+\varepsilon)\cdot OPT(\mathcal{I}_{t'}) + 5\varepsilon V_t + O(\varepsilon^{-2}).\end{aligned}$$

Now observe that $OPT(\mathcal{I}_{t'}) \geq V_t - \varepsilon V_t = (1-\varepsilon)\cdot V_t$ because the total volume of jobs at time t' is at least $V_t - \varepsilon V_t$, and so $OPT(\mathcal{I}_{t'}) \geq V_t/2$. Therefore, $5\varepsilon V_t \leq 10\varepsilon\, OPT(\mathcal{I}_{t'})$. It follows that the number of bins used by the algorithm at time t' is at most $(1+O(\varepsilon))\cdot OPT(\mathcal{I}_{t'}) + O(\varepsilon^{-2})$. ■

C.3.2 The Matching Lower Bound

Here we prove our matching lower bound for amortized recourse in the size costs setting.

We recall that our proof relies on Sylvester's sequence. For ease of reference we restate the salient properties of this sequence which we rely on in our proofs. The Sylvester sequence is given by the recurrence relation $k_1 = 2$ and $k_{i+1} = \left(\prod_{j\leq i} k_j\right) + 1$, or equivalently $k_{i+1} = k_i \cdot (k_i - 1) + 1$ for $i \geq 0$. The first few terms of this sequence are $2, 3, 7, 43, 1807, \ldots$ In what follows we let c be a large positive integer to be specified later, and $\varepsilon := 1/\prod_{\ell=1}^{c} k_\ell$. For notational simplicity, since our products and sums will always be taken over the range $[c]$ or $[c]\setminus\{i\}$ for some $i \in [c]$, we will write $\sum_\ell k_\ell$ and $\prod_\ell 1/k_\ell$, $\sum_{\ell\neq i} 1/k_\ell$, $\prod_{\ell\neq i} 1/k_\ell$, etc., taking the range of ℓ to be self-evident from context. In our proof of Theorem 5.6 and the lemmas building up to it, we shall make use of the following properties of the Sylvester sequence, its reciprocals and this $\epsilon = 1/\prod_\ell k_\ell$.

(P1) $\frac{1}{k_1} + \frac{1}{k_2} + \ldots + \frac{1}{k_c} = 1 - \frac{1}{\prod_\ell k_\ell} = 1 - \varepsilon$.

(P2) If $i \neq j$, then k_i and k_j are relatively prime.

(P3) For all $i \in [c]$, the value $1/k_i = \prod_{\ell\neq i} k_\ell / \prod_\ell k_\ell$ is an integer product of $\varepsilon = 1/\prod_\ell k_\ell$.

(P4) If $i \neq j \in [c]$, then $1/k_i = \prod_{\ell\neq i} k_\ell / \prod_\ell k_\ell$ is an integer product of $k_j \cdot \epsilon = k_j / \prod_\ell k_\ell$.

Properties 4 and 3 are immediate, while property 2 follows directly from the recursive definition of the sequence's terms, $k_{i+1} = \left(\prod_{j\leq i} k_j\right) + 1$. Finally, the same definition readily implies $\frac{1}{k_i} = \frac{1}{k_i - 1} - \frac{1}{k_{i+1}-1}$, from which property 1 follows by induction.

The instances we consider for our lower bound will be comprised of items with sizes given by entries of the following vector $\vec{s} \in [0,1]^{c+1}$, defined as follows:

$$s_i := \begin{cases} \frac{1}{k_i}\cdot\left(1 - \frac{\varepsilon}{2}\right) & i \in [c] \\ \varepsilon\cdot\left(\frac{3}{2} - \frac{\varepsilon}{2}\right) & i = c+1. \end{cases}$$

Let $\vec{1}$ be the all-ones vector, and $\vec{e}_i$ the i-th standard basis vector. In what follows we will denote by $\vec{\chi} \in \mathbb{N}^{c+1}$ a characteristic vector of a feasibly-packed bin; i.e., χ_i is the number of items of size s_i in the bin, and as the bin is not over-flowing, we have $\vec{\chi}\cdot\vec{s} = \sum_i \chi_i \cdot s_i \leq 1$. The following fact is easy to check from the definition of $\vec{s}$.

Fact C.3.1 $\vec{1} \cdot \vec{s} = 1$.

Proof: By property 1, we have $\sum_{i=1}^{c} s_i = (1 - \frac{\varepsilon}{2}) \cdot (1 - \varepsilon) = 1 - s_{c+1}$. ■

We now show that the item sizes given by the entries of $\vec{s}$ are such that in many situations (in particular, situations where no s_{c+1}-sized items are present), any feasible packing in a bin will leave some gap.

Lemma C.3.2 *Let χ denote a feasible packing of items into a bin; i.e., $\vec{\chi} \in \mathbb{N}^{c+1}$, $\vec{\chi} \cdot \vec{s} \leq 1$. Then,*
(a) If $\chi_{c+1} = 1$ and $\vec{\chi} \neq \vec{1}$, then $\vec{\chi} \cdot \vec{s} \leq 1 - \varepsilon/2$.
(b) If $\chi_{c+1} = 0$, then $\vec{\chi} \cdot \vec{s} \leq 1 - \varepsilon/2$.
(c) If $\chi_{c+1} = 0$ and $\vec{\chi} \neq k_i \vec{e}_i$, then $\vec{\chi} \cdot \vec{s} \leq 1 - \varepsilon$.

Proof: For ease of notation, we define a vector $\vec{t}$, where $t_i = 1/k_i$ for $i = 1, \ldots, c$ and $t_{c+1} = 0$. So, $s_i = (1 - \varepsilon/2) \cdot t_i$ for $i = 1, \ldots, c$.

Observation C.3.3 *The dot product $\vec{\chi} \cdot \vec{t}$ is at most 1.*

Proof: By property 3, each t_i is an integral multiple of ε. It follows that $\vec{\chi} \cdot \vec{t}$ is also an integral multiple of ε. Clearly, 1 is an integral multiple of ε. Therefore, if $\vec{\chi} \cdot \vec{t} > 1$, then $\vec{\chi} \cdot \vec{t}$ is at least $1 + \varepsilon$. But then, $\vec{\chi} \cdot \vec{s} \geq (1 - \varepsilon/2) \cdot \vec{\chi} \cdot \vec{t} \geq (1 - \varepsilon/2) \cdot (1 + \varepsilon) > 1$, a contradiction. ■

Part a: First, observe that $\vec{\chi} \cdot \vec{t} \neq 1$. Indeed, it is at most 1, by Observation C.3.3). Moreover, if $\vec{\chi} \cdot \vec{t} = 1$, the fact that $\chi_{c+1} = 1$ would imply $\vec{\chi} \cdot \vec{s} = \vec{\chi} \cdot (1 - \varepsilon/2) \cdot \vec{t} + s_{c+1} = (1 - \varepsilon/2) + s_{c+1} > 1$, a contradiction. Also, there must be some index $i \in [c]$ such that $\chi_i = 0$. Indeed, otherwise the fact that $\vec{\chi} \neq \vec{1}$ and Fact C.3.1 imply that $\vec{\chi} \cdot \vec{s} > \vec{1} \cdot \vec{s} = 1$. So let i be an index such that $\chi_i = 0$. By property 4, for all $j \neq i$, the value $t_j = 1/k_j$ is an integral multiple of $k_i\varepsilon$. Therefore, $\vec{\chi} \cdot \vec{t}$ is a multiple of $k_i\varepsilon$. Since $\vec{\chi} \cdot \vec{t} < 1$ and 1 is an integral multiple of $k_i\varepsilon$, it follows that $\vec{\chi} \cdot \vec{t} \leq 1 - k_i\varepsilon \leq 1 - 2\varepsilon$. Therefore, $\vec{\chi} \cdot \vec{s} \leq (1 - \varepsilon/2) \cdot (1 - 2\varepsilon) + s_{c+1} = 1 - \varepsilon + \varepsilon^2/2 \leq 1 - \varepsilon/2$.

Part b follows directly from Observation C.3.3 above. Indeed, since $\chi_{c+1} = 0$, we have that $\vec{\chi} \cdot \vec{s} = (1 - \varepsilon/2)\vec{\chi} \cdot \vec{t} \leq (1 - \varepsilon/2)$.

Part c: First, observe that $\chi_i < k_i$ for $i = 1, \ldots, c$. Indeed, if $\chi = k_i\vec{e}_i + \chi'$ for some index i and some non-zero non-negative vector χ', then $\vec{\chi} \cdot \vec{t} = 1 + \vec{\chi'} \cdot \vec{t} > 1$, which contradicts Observation C.3.3. Now let i be an index such that $\chi_i \geq 1$. Since $\chi_i < k_i$, property 2 implies that $\chi_i \cdot t_i = \chi_i \cdot \varepsilon \cdot \prod_{\ell \neq i} k_\ell$ is not an integral multiple of $k_i\varepsilon$. But, by property 4 for all $j \neq i$ we have that $t_j = 1/k_j$ is an integral multiple of $k_i\varepsilon$. Thus, $\vec{\chi} \cdot \vec{t} (\leq 1)$ is not an integral multiple of $k_i\varepsilon = k_i / \prod_\ell k_\ell = 1 / \prod_{\ell \neq i} k_\ell$. Consequently, $\vec{\chi} \cdot \vec{t} < 1$. But, by property 3, we have that $\vec{\chi} \cdot \vec{t}$ is an integral multiple of ε, from which it follows that $\vec{\chi} \cdot \vec{t} \leq 1 - \varepsilon$. Therefore, as $\chi_{c+1} = 0$, we have $\vec{\chi} \cdot \vec{s} = \vec{\chi} \cdot (1 - \epsilon/2) \cdot \vec{t} \leq (1 - \epsilon/2) \cdot (1 - \varepsilon) \leq 1 - \varepsilon$. ■

Equipped with the above lemma, we can show two instances with similar overall weight, $\mathcal{I}$ to $\mathcal{I}'$ for which near-optimal solutions different significantly. Specifically, we define the input instances $\mathcal{I}$ and $\mathcal{I}'$, as follows. For some large N a product of $\prod_\ell k_\ell$, Instance $\mathcal{I}$ consists of N items of sizes s_i for all $i \in [c + 1]$. Instance $\mathcal{I}'$ consists of N items of all sizes but s_{c+1}. It is easy to check that $c = \Theta(\log \log 1/\varepsilon)$, and so the total number of items is $n = \Theta(N \cdot \log \log(1/\varepsilon))$. Therefore the additive $o(n)$ term, denoted by $f(n)$, satisfies

$$f(n) < \epsilon \cdot n/(42 \cdot \Theta(\log \log(1/\varepsilon))) = \epsilon N/42.$$

We now proceed to proving that approximately-optimal packings of the above two instances differ significantly.

Any algorithm $\mathcal{A}$ with $(1+\varepsilon/7)$-a.c.r and $o(n)$ additive term packs instance $\mathcal{I}$ such that at least $2N/3$ bins contain exactly one item of each size s_i, and packs instance $\mathcal{I}'$ such that at least $N/2$ bins contain items of exactly one size.

Proof: We begin by proving our claimed bound on $\mathcal{A}$'s packing of $\mathcal{I}$. Fact C.3.1 shows that the optimal number of bins is N. Therefore, $\mathcal{A}$ is allowed to use at most $(1+\varepsilon/7)N + \varepsilon N/42 = (1+\varepsilon/6)N$ bins. Now suppose there more than $N/3$ bins for which the characteristic vector is not $\vec{1}$. Lemma C.3.2a shows that each such bin must leave out at least $\varepsilon/2$ space. Therefore, the total unused space in these bins is greater than $\varepsilon N/6$, which implies that the algorithm must use at least $N + \varepsilon N/6$ bins, a contradiction.

We now proceed to prove our claimed bound on $\mathcal{A}$'s packing of $\mathcal{I}'$. Let us first find the optimal value $OPT(\mathcal{I}')$. By property 1, the total volume of all the items is equal to $N(1-\varepsilon)(1-\varepsilon/2)$. By Lemma C.3.2b, any bin can be packed to an extent of at most $(1-\varepsilon/2)$. Therefore the optimal number of bins is at least $N(1-\varepsilon)$. Furthermore, we can achieve this bound by packing N items of size s_i in N/k_i bins for each index i. Therefore, the algorithm is allowed to use at most $(1+\varepsilon/7)(1-\varepsilon)N + \varepsilon N/42 \leq (1-\frac{35\varepsilon}{42})N$ bins when packing $\mathcal{I}'$.

Suppose there are at least $N/2$ bins each of which is assigned items of at least two different sizes by the algorithm. By Lemma C.3.2c, the algorithm will leave at least ε unused space in such bins; moreover, by Lemma C.3.2b, the algorithm will leave at least $\varepsilon/2$ unused space in every bin. Thus, the total unused space in the bins is at least $\varepsilon N/2 + \varepsilon N/4 = 3\varepsilon N/4$. Since the total volume of the items is equal to $N(1-\varepsilon)(1-\varepsilon/2)$, we see that the total number of bins used by the algorithm is at least $N(1-\varepsilon)(1-\varepsilon/2) + 3\varepsilon N/4 \geq N(1-3\varepsilon/4) > (1-\frac{35\varepsilon}{42})N$, a contradiction. ■

We are now ready to prove this section's main result. For infinitely many $\epsilon > 0$, any fully-dynamic bin packing algorithm with a.c.r $(1+\varepsilon)$ and additive term $o(n)$ must have *amortized* migration factor of $\Omega(\varepsilon^{-1})$.

Proof: We will show that any algorithm $\mathcal{A}$ using at most $(1+\varepsilon/7)\cdot OPT(\mathcal{I}_t) + o(n)$ bins at time t uses at least $1/160\varepsilon$ amortized recourse, for arbitrarily large optima and instance sizes, implying our theorem. We consider the two instance $\mathcal{I}$ and $\mathcal{I}'$ defined above.

Suppose we first provide instance $\mathcal{I}$ to $\mathcal{A}$, and then remove all items of size s_{c+1} to get the instance $\mathcal{I}'$. Let B_1 and B_2 be the sets of bins guaranteed by Section 5.6 when we had the instances $\mathcal{I}$ and $\mathcal{I}'$, respectively. Notice that as algorithm $\mathcal{A}$ uses at most $(1-\frac{35\varepsilon}{42})N \leq N$ bins while packing $\mathcal{I}'$, and $|B_1| \geq 2N/3$, $|B_2| \geq N/2$, we have that either (i) $|B_1 \cap B_2| \geq N/12$, or (ii) at least $N/12$ of the bins of B_1 are closed in $\mathcal{A}$'s packing of $\mathcal{I}'$.

Consider any bin which lies in both B_1 and B_2. In instance $\mathcal{I}$, algorithm $\mathcal{A}$ had assigned one item of each size to this bin, whereas in instance $\mathcal{I}'$ the algorithm assigns this bin items of only one size. Therefore, the total size of items which need to go out of (or into) this bin when we transition from $\mathcal{I}$ to $\mathcal{I}'$ is at least $1/2$. Therefore, the total volume of items moved during this transition is at least $N/48$. Similarly, if $N/12$ of the bins of B_1 are closed in $\mathcal{A}$'s packing of $\mathcal{I}'$, at least $1 - s_{c+1} \geq 1/2$ volume must leave each of these bins, and so the total volume of items moved during this transition is at least $N/24 > N/48$.

Repeatedly switching between $\mathcal{I}$ and $\mathcal{I}'$ by adding and removing the N items of size s_{c+1} a

total of T times (for sufficiently large T), we find that the amortized recourse is at least

$$\frac{T \cdot N/48}{N + T \cdot 3\varepsilon \cdot N} \geq \frac{1}{160\varepsilon}.$$

■

Bibliography

[1] Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1253–1264. SIAM, 2011. 6.1

[2] Miklós Ajtai, János Komlós, and Endre Szemerédi. A note on Ramsey numbers. *J. Comb. Theory, Ser. A*, 29(3):354–360, 1980. 2.2

[3] Miklós Ajtai, Paul Erdős, János Komlós, and Endre Szemerédi. On Turáns theorem for sparse graphs. *Combinatorica*, 1(4):313–317, 1981. 2.2

[4] Noga Alon. Independence numbers of locally sparse graphs and a Ramsey type problem. *Random Struct. Algorithms*, 9(3):271–278, 1996. 2.2, 2.3, 2.3, 2.3, 2.5, 2.5, 2.6.3, 2.6.4

[5] Noga Alon and Nabil Kahale. Approximating the independence number via the ϑ-function. *Math. Programming*, 80(3, Ser. A):253–264, 1998. ISSN 0025-5610. doi: 10.1007/BF01581168. URL `http://dx.doi.org/10.1007/BF01581168`. 2.2, 2.3, 2.4

[6] Noga Alon and Joel Spencer. *The Probabilistic Method*. Wiley Interscience, New York, 1992. 2.2

[7] Hyung-Chan An, Aditya Bhaskara, Chandra Chekuri, Shalmoli Gupta, Vivek Madan, and Ola Svensson. Centrality of trees for capacitated k-center. *Mathematical Programming*, 154 (1-2):29–53, 2015. 3.2

[8] Lachlan Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. A tale of two metrics: Simultaneous bounds on competitiveness and regret. In *Conference on Learning Theory*, pages 741–763, 2013. 7.2

[9] Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, Kevin Schewior, and Michele Scquizzato. Chasing convex bodies and functions. In *Latin American Symposium on Theoretical Informatics*, pages 68–81. Springer, 2016. 7.2

[10] C. J. Argue, Sébastien Bubeck, Michael B. Cohen, Anupam Gupta, and Yin Tat Lee. A nearly-linear bound for chasing nested convex bodies. *CoRR*, abs/1806.08865, 2018. URL `http://arxiv.org/abs/1806.08865`. 7.2

[11] Jonathan Aronson, Martin Dyer, Alan Frieze, and Stephen Suen. Randomized greedy matching. II. *Random Structures & Algorithms*, 6(1):55–73, 1995. 6.2

[12] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004. 3.2

[13] Per Austrin, Subhash Khot, and Muli Safra. Inapproximability of vertex cover and independent set in bounded degree graphs. *Theory of Computing*, 7(1):27–43, 2011. 2.2

[14] Maria-Florina Balcan, Florin Constantin, Satoru Iwata, and Lei Wang. Learning valuation functions. In *COLT*, volume 23, pages 4–1, 2012. 4.1

[15] Michel Louis Balinski. On finding integer solutions to linear programs. Technical report, DTIC Document, 1964. 3.4.2

[16] János Balogh, József Békési, Gábor Galambos, and Gerhard Reinelt. Lower bound for the online bin packing problem with restricted repacking. *SIAM Journal on Computing*, 38(1): 398–410, 2008. 5.2, 5.3, 5.4.1, 5.4.1

[17] János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. *Theoretical Computer Science*, 440:1–13, 2012. 5.1

[18] János Balogh, József Békési, Gábor Galambos, and Gerhard Reinelt. On-line bin packing with restricted repacking. *Journal of Combinatorial Optimization*, 27(1):115–131, 2014. 5.2

[19] Nikhil Bansal, Anupam Gupta, and Guru Guruganesh. On the Lovász theta function for independent sets in sparse graphs. *CoRR*, abs/1504.04767, 2015. URL `http://arxiv.org/abs/1504.04767`. 2.2, A.1

[20] Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Kirk Pruhs, Kevin Schewior, and Cliff Stein. A 2-competitive algorithm for online convex optimization with switching costs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 40. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015. 7.2

[21] Nikhil Bansal, Martin Böhm, Marek Eliáš, Grigorios Koumoutsos, and Seeun William Umboh. Nested convex bodies are chaseable. *arXiv preprint arXiv:1707.05527*, 2017. 7.2

[22] Yair Bartal, Moses Charikar, and Danny Raz. Approximating min-sum k-clustering in metric spaces. In *Proceedings of the 33rd STOC*, pages 11–20, 2001. 3.2, 3.5

[23] Babak Behsaz, Zachary Friggstad, Mohammad R. Salavatipour, and Rohit Sivakumar. Approximation algorithms for min-sum k-clustering and balanced k-median. In *Proceedings of the 42nd ICALP*, pages 116–128, 2015. 3.2

[24] W. Ben-Ameur and H. Kerivin. Routing of uncertain demands. *Optimization and Engineering*, 3:283–313, 2005. 4.2

[25] Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. Fully dynamic bin packing revisited. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, pages 135–151, 2015. URL `http://dx.doi.org/10.4230/LIPIcs.APPROX-RANDOM.2015.135`. 5.2, 5.3, 5.3, 5.4.2, 5.6, C.1.2

[26] Kshipra Bhawalkar and Tim Roughgarden. Welfare guarantees for combinatorial auctions

with item bidding. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 700–709. Society for Industrial and Applied Mathematics, 2011. 4.1

[27] Tom Bohman and Peter Keevash. The early evolution of the H-free process. *Inventiones Mathematicae*, 181(2):291–336, 2010. 2.5.1

[28] Allan Borodin, Nathan Linial, and Michael E Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM (JACM)*, 39(4):745–763, 1992. 1.2.3, 7.2

[29] Niv Buchbinder, Joseph Seffi Naor, et al. The design of competitive online algorithms via a primal–dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3): 93–263, 2009. 7.2

[30] Jaroslaw Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the 26th SODA*, pages 737–756, 2015. 3.2

[31] D. Chakrabarty and C. Swamy. Facility location with client latencies: Lp-based techniques for minimum-latency problems. *Math. Oper. Res.*, 41(3):865–883, 2016. 4.3, 4.6, 4.6, 4.6

[32] Siu On Chan. Approximation resistance from pairwise independent subgroups. In *STOC*, pages 447–456, 2013. doi: 10.1145/2488608.2488665. URL `http://doi.acm.org/10.1145/2488608.2488665`. 2.2

[33] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM Journal on Computing*, 34(4):803–824, 2005. 3.2

[34] Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *Journal of Computer and System Sciences*, 68(2):417–441, 2004. 3.2

[35] Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002. 3.2

[36] C. Chekuri. Routing and network design with robustness to changing or uncertain traffic demands. *SIGACT News*, 38(3):106–128, 2007. 4.2

[37] Chandra Chekuri and Kent Quanrud. Fast approximations for matroid intersection. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, 2016. 6.2, 6.3

[38] Eden Chlamtac and Madhur Tulsiani. Convex relaxations and integrality gaps. In Miguel F. Anjos and Jean B. Lasserre, editors, *Handbook on Semidefinite, Conic and Polynomial Optimization*. Springer, 2012. 2.4, 2.4.2

[39] M. Chlebik and J. Chlebikova. Approximation hardness of the steiner tree problem on graphs. *Proceedings of the Scandinavian Workshop on Algorithm Theory*, pages 170–170, 2002. 4.1

[40] Julia Chuzhoy and Yuval Rabani. Approximating k-median with non-uniform capacities. In *Proceedings of the 16th SODA*, pages 952–958, 2005. 3.2, 3.5

[41] Edward G Coffman Jr, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: survey and classification. In *Handbook of Combinatorial Optimization*, pages 455–531. Springer, 2013. 5.1

[42] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971. 1

[43] Robert M Corless, Gaston H Gonnet, D EG Hare, David J Jeffrey, and Donald E Knuth. On the Lambert-W function. *Advances in Computational mathematics*, 5(1):329–359, 1996. 5.3

[44] Marek Cygan. Improved approximation for 3-dimensional matching via bounded pathwidth local search. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 509–518. IEEE, 2013. 2.7

[45] Marek Cygan, MohammadTaghi Hajiaghayi, and Samir Khuller. LP rounding for k-centers with non-uniform hard capacities. In *Proceedings of the 53rd FOCS*, pages 273–282, 2012. 3.2

[46] W Fernandez De La Vega and George S. Lueker. Bin packing can be solved within 1+ ε in linear time. *Combinatorica*, 1(4):349–355, 1981. C.1.2, C.3.1

[47] Nikhil R Devanur, Kamal Jain, and Robert D Kleinberg. Randomized primal-dual analysis of ranking for online bipartite matching. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 101–107, 2013. 6.1

[48] N.G. Duffield, P. Goyal, A.G. Greenberg, P.P. Mishra, K.K. Ramakrishnan, and J.E. van der Merwe. A flexible model for resource management in virtual private networks. *Proceedings of SIGCOMM*, 29:95–108, 1999. 4.2

[49] Martin Dyer and Alan Frieze. Randomized greedy matching. *Random Structures & Algorithms*, 2(1):29–45, 1991. B.2.2

[50] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69(125-130):55–56, 1965. 1

[51] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. *Combinatorial structures and their applications*, pages 69–87, 1970. 6.2

[52] Leah Epstein and Asaf Levin. A robust APTAS for the classical bin packing problem. *Math. Program.*, 119(1):33–49, 2009. doi: 10.1007/s10107-007-0200-y. URL `http://dx.doi.org/10.1007/s10107-007-0200-y`. 5.2, 5.3, 5.6

[53] Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011. 6.2

[54] Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935. 2.5

[55] Paul Erdos and Alfred Renyi. On random matrices. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 8(455-461):1964, 1964. B.2.4

[56] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69:485–497, 2004. 4.1

[57] Uriel Feige. Randomized graph products, chromatic numbers, and the lovász ϑ-function. *Combinatorica*, 17(1):79–90, 1997. 2.7

[58] Uriel Feige. Approximating maximum clique by removing subgraphs. *SIAM J. Discrete Math.*, 18(2):219–225, 2004. 2.2

[59] Uriel Feige. On maximizing welfare when utility functions are subadditive. *SIAM Journal on Computing*, 39(1):122–142, 2009. 4.1

[60] Uriel Feige and Anne Kenyon. On the profile of multiplicities of complete subgraphs. *arXiv preprint arXiv:1703.09682*, 2017. 2.7

[61] B. Feldkord, M. Feldotto, and S. Riechers. A Tight Approximation for Fully Dynamic Bin Packing without Bundling. *ArXiv e-prints*, November 2017. 5.7

[62] A.E. Feldmann, J. Könemann, K. Pashkovich, and L. Sanità. Fast approximation algorithms for the generalized survivable network design problem. *Proceedings of ISAAC (International symposium on algorithms and computation)*, pages 33:1– 33:12, 2016. 4.2

[63] J. Fingerhut, S. Suri, and J. Turner. Designing least-cost nonblocking broadband networks. *J. Algorithms*, 24(2):287–309, 1997. 4.2

[64] Joel Friedman and Nathan Linial. On convex body chasing. *Discrete & Computational Geometry*, 9(1):293–321, 1993. 1.2.3, 7.2

[65] Giorgio Gambosi, Alberto Postiglione, and Maurizio Talamo. New algorithms for on-line bin packing. In *Algorithms and Complexity, Proceedings of the First Italian Conference*, pages 44–59, 1990. 5.2

[66] Giorgio Gambosi, Alberto Postiglione, and Maurizio Talamo. Algorithms for the relaxed online bin-packing model. *SIAM journal on computing*, 30(5):1532–1551, 2000. 5.2

[67] Bernd Gärtner and Jiří Matoušek. *Approximation algorithms and semidefinite programming*. Springer, Heidelberg, 2012. ISBN 978-3-642-22014-2; 978-3-642-22015-9. doi: 10.1007/978-3-642-22015-9. URL `http://dx.doi.org/10.1007/978-3-642-22015-9`. 2.4

[68] Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 982–991. Society for Industrial and Applied Mathematics, 2008. 6.1, 6.2

[69] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42 (6):1115–1145, November 1995. ISSN 0004-5411. doi: 10.1145/227683.227684. URL `http://doi.acm.org/10.1145/227683.227684`. 1.1

[70] M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995. 4.1, 4.2

[71] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. 3.2

[72] N. Goyal, N. Olver, and F. B. Shepherd. Dynamic vs. oblivious routing in network design. *Algorithmica*, 61(1):161–173, 2011. 4.3

[73] N. Goyal, N. Olver, and F. B. Shepherd. The vpn conjecture is true. *J. ACM*, 60(3):17:1–17:17, June 2013. ISSN 0004-5411. doi: 10.1145/2487241.2487243. URL `http://doi.acm.org/10.1145/2487241.2487243`. 4.2

[74] F. Grandoni, T. Rothvoß, and L. Sanità. From uncertainty to non-linearity: Solving virtual private network via single-sink buy-at-bulk. *Math. Oper. Res.*, 36(2):185–204, 2011. 4.2

[75] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*. Springer-Verlag, Berlin, 1988. 2.4

[76] A. Gupta, J. Kleingerg, R. Kumar, B. Rastogi, and B. Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. *Proceedings of Sympos. Theory Comput. (STOC)*, pages 389–398, 2001. 4.2

[77] A. Gupta, V. Nagarajan, and R. Ravi. An improved approximation algorithm for requirement cut. *Operations Research Letters*, 38(4):322–325, 2010. 4.1

[78] Magnús M. Halldórsson. Approximations of independent sets in graphs. In *APPROX*, pages 1–13, 1998. 2.2

[79] Magnús M. Halldórsson. Approximations of weighted independent set and hereditary subset problems. *J. Graph Algorithms Appl.*, 4:no. 1, 16 pp., 2000. ISSN 1526-1719. doi: 10.7155/jgaa.00020. URL `http://dx.doi.org/10.7155/jgaa.00020`. 2.2, 2.4

[80] Magnús M. Halldórsson and Jaikumar Radhakrishnan. Improved approximations of independent sets in bounded-degree graphs via subgraph removal. *Nord. J. Comput.*, 1(4):475–492, 1994. 2.2

[81] Eran Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM J. Comput.*, 31(5):1608–1623, 2002. 2.2, 2.3, 2.4, 2.4.1, 2.4

[82] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *FOCS*, pages 627–636, 1996. 2.2

[83] Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-dimensional matching. In *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, pages 83–97. Springer, 2003. 2.7

[84] Sandy Heydrich and Rob van Stee. Beating the harmonic lower bound for online bin packing. In *43rd International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl, 2016. 5.1, 5.3

[85] Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2625. SIAM, 2017. 5.1

[86] Dorit S Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co.,

1996. 5.1

[87] Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10:180–184, 1985. 3.2

[88] John E Hopcroft and Richard M Karp. An $n^{5/2} algorithm for maximum matchings in bipartite graphs. SIA (4) : 225 - -231, 1973. 6.2$

[89] Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1:209–215, 1979. 3.2

[90] Chien-Chung Huang, Naonori Kakimura, and Naoyuki Kamiyama. Exact and Approximation Algorithms for Weighted Matroid Intersection. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2016. 6.2, 6.3

[91] Zoran Ivković and Errol L Lloyd. A fundamental restriction on fully dynamic maintenance of bin packing. *Information Processing Letters*, 59(4):229–232, 1996. 5.2

[92] K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001. 4.1, 4.2, 4.3

[93] 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. 3.2, 3.4.3, 3.5

[94] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the 34th STOC*, pages 731–740, 2002. 3.2

[95] Klaus Jansen and Kim-Manuel Klein. A robust AFPTAS for online bin packing with polynomial migration,. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 589–600, 2013. URL `http://dx.doi.org/10.1007/978-3-642-39206-1_50`. 5.2, 5.3, 5.4.2, 5.6

[96] Anders Johansson. The choice number of sparse graphs. preprint, August 1996. 2.2, 2.3, A.1

[97] Anders Johansson. Asymptotic choice number for triangle-free graphs. preprint, 1996. 2.2

[98] David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974. 5.1

[99] Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, pages 587–596. ACM, 2011. 6.2

[100] David R. Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45(2):246–265, 1998. 2.2, A.2.1

[101] Narendra Karmarkar and Richard M Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 312–320. IEEE Computer Society, 1982. C.3.1

[102] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972. 3.4.1

[103] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972. 1

[104] Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 352–358, 1990. 1.2.1, 6.1

[105] Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for MaxClique, chromatic number and Min-3Lin-Deletion. In *ICALP (1)*, pages 226–237, 2006. 2.2

[106] Jeong Han Kim. On Brooks' theorem for sparse graphs. *Combinatorics, Probability & Computing*, 4:97–132, 1995. 2.2

[107] Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 231–242. Springer, 2012. 6.2, 6.3, 6.4.3, 6.4.2, 6.5.2, 6.7.1, B.4

[108] Bernhard Korte and Jens Vygen. *Combinatorial Optimization, Volume 21 of Algorithms and Combinatorics*. Springer-Verlag, Berlin,, 2008. 6.2, 6.5.1

[109] Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *International Colloquium on Automata, Languages and Programming*, pages 508–520. Springer, 2009. 6.2

[110] Nitish Korula, Vahab Mirrokni, and Morteza Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 889–898. ACM, 2015. 6.1, 6.2

[111] Christian Kroer and Tuomas Sandholm. Extensive-Form Game Imperfect-Recall Abstractions With Bounds. *CoRR*, abs/1409.3302, 2014. also published at the Algorithmic Game Theory workshop at IJCAI, 2015. 3.1

[112] Monique Laurent. A comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for 0-1 programming. *Math. Oper. Res.*, 28(3):470–496, 2003. 2.4

[113] Monique Laurent. Networks and semidefinite programming (lecture notes 2014), 2014. http://homepages.cwi.nl/ monique/lnmb14/lnmb14.pdf. 2.4

[114] Chan C Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)*, 32(3):562–572, 1985. 5.1, 5.5.2, 5.5.2

[115] Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 18–28. ACM, 2001. 4.1

[116] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the 45th STOC*, pages 901–910, 2013. 3.2

[117] Minghong Lin, Adam Wierman, Alan Roytman, Adam Meyerson, and Lachlan LH Andrew. Online optimization with switching cost. *ACM SIGMETRICS Performance Evaluation Review*, 40(3): 98–100, 2012. 7.2

[118] László Lovász. On the Shannon capacity of a graph. *IEEE Trans. Inform. Theory*, 25(1):1–7, 1979.

ISSN 0018-9448. doi: 10.1109/TIT.1979.1055985. URL http://dx.doi.org/10.1109/TIT.1979.1055985. 2.1

[119] Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, pages 597–606, 2011. 6.2

[120] Aranyak Mehta. Online matching and ad allocation. *Theoretical Computer Science*, 8(4):265–368, 2012. 6.2

[121] Aranyak Mehta and Vijay Vazirani. Personal communication, 2015. 6.2

[122] Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22, 2007. 6.1

[123] Michael Molloy. The list chromatic number of graphs with small clique number. *arXiv preprint arXiv:1701.09133*, 2017. 2.7

[124] Marcin" "Mucha and "Smulewicz Marcin". "personal communication", 2018. 4.7

[125] G. Oriolo, L. Sanità, and R. Zenklusen. Network design with a discrete set of traffic matrices. *Operations Research Letters*, 41(4):390–396, 2013. 4.1, 4.2, 4.3

[126] James G Oxley. *Matroid Theory*, volume 3. Oxford university press, 2006. 6.5.1

[127] Prakash Ramanan, Donna J Brown, Chung-Chieh Lee, and Der-Tsai Lee. On-line bin packing in linear time. *Journal of Algorithms*, 10(3):305–326, 1989. 5.1

[128] Michael B Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34(1-3):203–227, 1991. 5.1

[129] Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009. 5.2

[130] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2002. 6.2

[131] Steven S Seiden. On the online bin packing problem. *Journal of the ACM (JACM)*, 49(5):640–671, 2002. 5.1, 5.3.2, 5.5.2, 5.5.2, C.2.2, C.2.2

[132] James B. Shearer. A note on the independence number of triangle-free graphs. *Discrete Math.*, 46 (1):83–87, 1983. ISSN 0012-365X. doi: 10.1016/0012-365X(83)90273-X. URL http://dx.doi.org/10.1016/0012-365X(83)90273-X. 2.2

[133] James B. Shearer. On the independence number of sparse graphs. *Random Structures Algorithms*, 7(3):269–271, 1995. ISSN 1042-9832. doi: 10.1002/rsa.3240070305. URL http://dx.doi.org/10.1002/rsa.3240070305. 2.2, 2.3, 2.3, 2.5

[134] Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. 1

[135] James J Sylvester. On a point in the theory of vulgar fractions. *American Journal of Mathematics*, 3(4):332–335, 1880. 5.6

[136] JD Ullman. *The performance of a memory allocation algorithm*. Princeton University, 1971. 5.1

[137] André van Vliet. *Lower and Upper Bounds for On-line Bin Packing and Scheduling Heuristics: Onder-en Bovengrenzen Voor On-line Bin Packing en Scheduling Heuristieken*. Thesis Publ., 1995. 5.1

[138] Yajun Wang and Sam Chiu-wai Wong. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *International Colloquium on Automata, Languages, and Programming*, pages 1070–1081. Springer, 2015. 6.1

[139] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. Available at `http://www.designofapproxalgs.com`. 1, 3.2

[140] Gerhard Woeginger. Improved space for bounded-space, on-line bin-packing. *SIAM Journal on Discrete Mathematics*, 6(4):575–581, 1993. 5.1

[141] M. Yannakakis. Node-and edge-deletion np-complete problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 253–264. ACM, 1978. 4.5

[142] Andrew Chi-Chih Yao. New algorithms for bin packing. *Journal of the ACM (JACM)*, 27(2): 207–227, 1980. 5.1

[143] M. Zadimoghaddam. Online Weighted Matching: Beating the $\frac{1}{2}$ Barrier. *ArXiv e-prints*, April 2017. 6.8