Scheduling Algorithms for Parallel Processing and Buffering Problems

by Hermann Wilhelm Richard Stotz

Technische Universität München
Fakultät für Informatik

Fakultät für Informatik
Lehrstuhl für Algorithmen und Komplexität

Scheduling Algorithms for Parallel Processing and Buffering Problems

Hermann Wilhelm Richard Stotz

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Michael G. Bader
Prüfer der Dissertation:	1. Prof. Dr. Harald Räcke
	2. Prof. Dr. Heiko Röglin

Die Dissertation wurde am 24.02.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 08.07.2020 angenommen.

Abstract

Scheduling, the task of sorting and distributing workload, is a fundamental area of algorithmic research. In this work, we analyze scheduling problems in two elementary scenarios.

In the first part of the thesis, we are interested in the problem of partitioning the vertex set of a hypergraph into parts of equal size. It is our goal to minimize the number of hyperedges between parts. This problem derives its importance from a scheduling task arising in high-performance computing. The processes of a parallel application are to be mapped onto processors in a parallel system so that on the one hand all processors receive the same workload and on the other hand the necessary communication between processors is minimal. Balanced partitioning of simple graphs is a classical and well-studied problem whereas only limited theoretical results are known for the hypergraph case. We shed new light on this scenario by discussing algorithmic strategies for approaching it and developing approximation algorithms for several of its variants. By proving strong lower bounds on the approximation factors achievable in polynomial time, we also establish a strong distinction between partitioning problems in graphs and hypergraphs.

In the second part of the thesis, we study the Generalized Reordering Buffer Management problem. Buffer management problems arise in scheduling scenarios where the execution of a task can be delayed in order to improve the overall performance of the system. More formally, we consider a sequence of colored items arriving online. Each item must be processed by one of k servers. In each time step, the scheduling strategy must choose an item from the b still unprocessed items and a server to process that item. Processing an item only incurs cost if the previously processed item of that server had a different color. The goal is to minimize the total cost incurred by the algorithm. We present an algorithm that achieves competitive ratio $\mathcal{O}(\log k(\log k + \log\log b))$ for Generalized Reordering Buffer Management with uniform costs. Our result is a considerable improvement over previous work and the algorithm is asymptotically optimal if k is a fixed constant.

Acknowledgements

First and foremost, I want to thank my advisor Harald Räcke for his guidance and support throughout my studies. He always had an open door to discuss everything, from a classic result to the latest papers, and I am immensely grateful for his trust and encouragement. I also want to thank Heiko Röglin for agreeing to review my thesis and Michael Bader for taking the time to hold the chair of the examination committee.

I want to thank Matthias Englert and Roy Schwartz for many inspiring discussions during their respective stays in Munich.

My colleagues at the chair of Algorithms and Complexity provided a very enjoyable and productive working atmosphere. Our monthly *PhDinners* were a highlight of my time at the university! Special thanks to Luisa Peter and Maximilian Janke for proofreading parts of this thesis, Leon Ladewig for verifying a last-minute proof and Dennis Kraft for many insightful conversations.

I would especially like to thank my family. My wife Eva-Maria and our children always helped and supported me with their encouragement, love and patience. My parents and my sister deserve special thanks for their continued support and encouragement.

Contents

1. Introduction

Whenever an assignment is so complex that it must be split into multiple subtasks, we open the box of Pandora that is scheduling, the burden of arranging and dividing that workload. But just as Pandora's box ultimately contains the virtue of Hope, so lies in scheduling one of the most rich and fruitful directions of algorithmic research. In less poetic terms, scheduling is the duty of distributing and ordering tasks with the goal of fulfilling some measure of efficiency. Very often a single decision maker must find a strategy to arrange the workload subject to multiple or even conflicting constraints. The performance of the strategy can then be analyzed using formal mathematical analysis, experimental study or a combination of the two. This work focuses on the mathematical analysis of two fundamental scenarios from the vast landscape of scheduling problems.

The first part of this thesis is devoted to algorithms for load balancing in parallel systems. A classical challenge when using highly parallel machines is the distribution of the workload among the machine's processors; see Codd for a very early discussion [Cod60]. In modern large scale scientific computing, two major goals have been identified for the scheduling algorithm: First, the load on a processor should be proportional to its speed. Second, the amount of communication between processors should be minimized. It is therefore convenient to model the workload as a graph whose vertices represent the units of computation and whose edges represent the communication requirement between units of computation. This leads to the classical algorithmic area that is graph partitioning. A typical graph partitioning problem asks, for example, to divide the vertex set of the graph into roughly equal-sized parts (corresponding to identical processors) so that the number of edges between parts (corresponding to the communication requirement) is minimized.

While the above model has been widely used in both theory and practice, researchers have long raised the need for a more expressive representation of communication than graphs, see, e.g., Hendrickson and Kolda [HK00]. By representing the computation with a *hypergraph*, one can model scenarios such as multicast communication more accurately. In a hypergraph, the vertices are connected through *hyperedges* that encompass *at least* two vertices. Standard graphs are the special case of hypergraphs in which every hyperedge encompasses *exactly* two vertices. Hypergraph partitioning is therefore a natural extension

of graph partitioning. The use of hypergraphs instead of graphs may however come at the cost of increased algorithmic complexity. We will develop strategies for computing balanced partitions in graphs and hypergraphs and give an overview of the computational hardness of hypergraph partitioning problems.

In its second part, this thesis studies algorithms for scheduling a sequence of tasks that arrive one-by-one. We are interested in scenarios where tasks can be delayed in order to reduce the cost of processing the sequence. As an example by Spieckermann et al., consider a painting shop in a car manufacturing plant [SGV04]. Painting two consecutive cars with different colors incurs significant setup costs compared to painting them with the same color through, e.g., cleaning procedures. The overall cost of processing the sequence is hence dominated by the number of color changes, i.e., the number of *context switches* of the service station. In this scenario it may be possible to delay some tasks of the sequence, for example by temporarily storing some cars in a parking lot. This allows the scheduling strategy to partially reorder the sequence. If, for instance, the sequence is *red*, *blue*, *red*, delaying the *blue* car to the end of the sequence reduces the number of context switches by one. Hence, the reordered sequence incurs a smaller total cost. A popular method of allowing limited reordering power is to employ a *reordering buffer* of limited capacity. The scheduling strategy can either decide to process the next task of the sequence or place it in the buffer instead. When the buffer fills up, the strategy must pick a task from the buffer to process.

The basic scenario outlined above can be varied and extended in multiple ways, some of which are outlined next. We say that tasks arrive *online* if the scheduling strategy has to make its decisions with no knowledge about future requests. Furthermore, the cost of a context switch might depend on both the old and the new state. In the Generalized Reordering Buffer Management problem (GRBM), there are multiple service stations available to process a task. In the example of the car manufacturing plant, this corresponds to multiple painting stations built in order to reduce the number of necessary color changes. We give a detailed analysis of GRBM and we present an algorithm for its online variant.

In the remainder of this introductory chapter, we give a more detailed introduction to the two main areas of this thesis along with a formal definition of the problems we consider. We also review the state of the art and discuss related work. Finally, we give a summary of the new results we obtained in this work.

1.1. Balanced Cuts in Graphs and Hypergraphs

In balanced graph partitioning, one is asked to partition the vertex set of a graph into parts of equal or similar size, so as to minimize the *boundary* of the parts. Most often, the boundary of a part is defined as the number of edges with exactly one endpoint in the part. Alternatively, the number of vertices connected to these edges, the number of directed edges leaving the part, or, if the underlying structure is a hypergraph, the number of hyperedges with some but not all of its vertices within the part may be counted. As argued above, this representation has been used successfully to model the task of load balancing in parallel systems. We will, however, use the graph partitioning terminology to keep the presentation in line with the majority of the literature.

As an exemplary problem, we consider the task of partitioning a (hyper-)graph of n vertices into two parts A and B. Given a partition, let the *size* of this *cut* be the number of (hyper-)edges with endpoints in both A and B.

If A and B are allowed to be of arbitrary size, the task is to find a minimum cut in a graph. This classical Global Mincut problem is solvable in polynomial time, even for hypergraphs, via a reduction to flow networks, see, e.g., Chekuri and Xu [CX17] and references therein. For the case when A and B should have roughly the same size, the Min-Ratio Cut problem has been a very influential ambassador. In this problem, the goal is find a cut (A, B) that minimizes the ratio of the size of the cut $\delta(A, B)$ to the size of the smaller part $\min\{|A|, |B|\}$, i.e., the goal is to minimize $\frac{\delta(A,B)}{\min\{|A|,|B|\}}$. Min-Ratio Cut is NP-hard, as shown by Matula and Shahrokhi [MS90], hence it likely cannot be solved with a polynomial-time algorithm. In a highly influential work, Leighton and Rao gave an approximation algorithm to find a cut of ratio at most $\mathcal{O}(\log n)$ of the optimum [LR99]. They also demonstrate how to use an algorithm for finding min-ratio cuts as a subroutine in many other balanced graph partitioning algorithms. Most of these results have been improved using more advanced techniques introduced by Arora et al. [ARV09], e.g., there is an algorithm for approximating the min-ratio cut up to a factor of $\mathcal{O}(\sqrt{\log n})$; see further discussion in Section 1.1.2.

The main algorithm developed by Leighton and Rao is based on first defining a metric on the given graph. This metric is then used in order to find sets of cut cost proportional to the *volume* of the set in the metric, i.e., the length of edges times their cost. One can show that there always exists such a set, and that the total volume of the graph is a lower bound on the cost of an optimal min-ratio cut. Leighton and Rao also apply their technique to directed graphs. Via reductions, this leads to $\mathcal{O}(\log n)$-approximations for Min-Ratio Cut in hypergraphs.

When dealing with exact balance constraints, different techniques are necessary. This is well-illustrated by the connection of Minimum Bisection to the classical Partition problem. Minimum Bisection asks to partition the vertex set of a graph into two parts of equal size so that the number of edges between parts is minimized. The problem is NP-hard, as shown by Garey et al. [GJS76]. Partition is a weakly NP-complete problem that asks to partition a multiset of integers into two sets so that both sum to the same value (Problem (SP12) as described by Garey and Johnson [GJ79]). In a graph of disjoint cliques, the Minimum Bisection problem essentially asks to solve Partition in order to decide whether to cut a single edge or not. Hence, even an approximation algorithm for Minimum Bisection requires a very careful control of the cuts generated by the algorithm.

The most successful approach to approximating the optimal cut size of the minimum bisection comes from the use of trees that are *edge cut sparsifiers*. These are constructions of weighted trees that come with a mapping of the tree vertices into the vertices of the original graph; while not entirely accurate, one can think of these trees as weighted spanning trees of the original graph. The goal of the mapping is that for any two sets of vertices A, B, the sizes for A and B are similar in the graph and in the tree. Perhaps surprisingly, it is possible to construct tree so that $\delta(A, B)$ is distorted by at most a polylogarithmic factor for any sets A, B. Räcke gives a construction of a family of such trees with expected distortion $\mathcal{O}(\log n)$ [Räc08]. Furthermore, he shows that a comparatively simple dynamic program suffices to compute the optimal bisection of a weighted tree. This gives an $\mathcal{O}(\log n)$-approximation algorithm for Minimum Bisection.

For problems which do not require strict balance, results for undirected graphs mostly extend to hypergraphs. In our work we investigate whether the same is true for problems with strict balance constraints such as Minimum Bisection. We are particularly interested in constructing trees that are cut sparsifiers for hypergraphs and vertex cuts, as those can be used to tackle a variety of partitioning problems.

1.1.1. Formal description of the problems

A hypergraph $G = (V, E)$ consists of a set of vertices V and a set of hyperedges E, with each hyperedge $e \in E$ being a subset of vertices of cardinality at least 2. If all hyperedges have the same size r, we say the hypergraph is r-uniform.

Given an hypergraph $G = (V, E)$ and disjoint vertex sets $A, B \subset V$, an (A, B) *edge cut* is a set of (hyper-)edges $F \subseteq E$ whose removal from E disconnects A and B. Let the *cost* (or *size*) $\delta_G(A, B)$ of the cut be the number of the edges in the smallest (A, B) edge cut. We write $\delta_G(A)$ for $\delta_G(A, V \setminus A)$ and drop the subscript if the hypergraph is

clear from the context. If $A \subset V$ has size $|A| = \lfloor n/2 \rfloor$, we say that A is a *bisection* of G with cost $\delta(A)$. The *sparsity* of a subset $\emptyset \neq A \subset V$ with $|A| < n/2$ is $\delta_G(A)/|A|$, a set of minimum sparsity is called *min-ratio cut* of G. We will sometimes consider hypergraphs with positive edge weights to which these definition extend naturally.

In the Minimum Bisection problem we are given an undirected graph G with the goal to find a bisection of G with smallest cost. Similarly, in the Minimum Hypergraph Bisection problem we are given a hypergraph G and are asked to find the bisection of G with smallest cost. An *α-approximation algorithm* for these problems is an algorithm that finds a bisection of cost at most $\alpha \cdot \mathrm{OPT}$, where OPT denotes the cost of an optimal bisection. We require that the running time of any approximation algorithm be polynomial in the size of the input.

Similarly, a *vertex cut* separating two disjoint sets of vertices A and B of an undirected graph $G = (V, E)$ is a set $X \subseteq V$ whose removal from V disconnects $A \setminus X$ and $B \setminus X$. The vertex cut may include vertices from both A and B. Let the cost $\gamma_G(A, B)$ of the vertex cut be the total number of vertices in a minimum-weight (A, B) vertex cut. Again, we may drop the subscript if no confusion is possible. If $A \subset V$ has size $\lfloor n/2 \rfloor$, we say that A is a *vertex bisection* of G with cost $\gamma(A, V \setminus A)$. In the Minimum Vertex Bisection problem we are given an undirected graph G and are asked to find a vertex bisection of G with smallest cost. We will sometimes consider graphs with positive vertex weights to which these definition naturally extend.

A *Vertex Separator* (A, B, X) is a set $X \subseteq V$ whose removal separates G into two disconnected pieces A and B (we do not require that A and B are connected themselves). Its *sparsity* is defined as

$$\frac{|X|}{\min\{|A|, |B|\} + |X|} \ .$$

A vertex separator of minimum sparsity is called *min-ratio vertex cut* of G.

For simplicity of exposition, we assume from now on that the number n of vertices of any hypergraph is even, so that both sides of a bisection have size $n/2$. This assumption does not affect our asymptotic results.

An *edge cut tree* $T = (V_T, E_T, w_T)$ of a hypergraph $G = (V, E)$ is an edge-weighted tree with $V \subseteq V_T$; typically, the vertices of G are the leaves of the tree. The edge cut tree is *dominating* if for any two disjoint sets $A, B \subset V$, we have $\delta_G(A, B) \leq \delta_T(A, B)$. The edge cut tree has *quality* or *approximation guarantee* α if for all disjoint $A, B \subset V$, it holds that

$$\delta_G(A, B) \leq \delta_T(A, B) \leq \alpha \cdot \delta_G(A, B) \ .$$

Observe that this implies that the tree is dominating.

Similarly, a *vertex cut tree* $T = (V_T, E_T, w_T)$ of a hypergraph $G = (V, E)$ is a vertex-weighted tree with $V \subseteq V_T$. The cut tree approximates *edge cuts* with quality α if for any two disjoint sets $A, B \subset V$,

$$\delta_G(A, B) \leq \gamma_T(A, B) \leq \alpha \cdot \delta_G(A, B) \ .$$

The tree approximates *vertex cuts* with quality α if for any two disjoint sets $A, B \subset V$,

$$\gamma_G(A, B) \leq \gamma_T(A, B) \leq \alpha \cdot \gamma_G(A, B) \ .$$

Conjectures and hypotheses in computational complexity

As is common practice, the hardness results in this thesis depend on established conjectures and hypotheses from computational complexity. Here, we only state those most relevant to our results. An in-depth introduction into the field is provided by the textbook by Arora and Barak [AB09]. For an entertaining perspective on the likelihood of some of these conjectures, see a recent survey by Williams [Wil19].

Certainly the most famous conjecture from the field is P $\neq$ NP, see Garey and Johnson [GJ79]. Here P denotes the set of decision problems that can be solved by a *deterministic Turing machine* in time polynomial in the input size. The set NP consists of the decision problems that can be solved by a *nondeterministic Turing machine* in time polynomial in the input size. Equivalently, the P $\neq$ NP conjecture states that there exists no polynomial-time algorithm for 3SAT. The 3SAT problem that asks to determine whether a given 3SAT formula is satisfiable.

The *Exponential Time Hypothesis* (ETH) is a stronger hypothesis formulated by Impagliazzo and Paturi.

Hypothesis 1.1 (ETH, [IP01])**.** 3SAT *on n variables cannot be solved in $2^{\varepsilon n}$ time, for some $\varepsilon > 0$.*

The *Hypergraph Dense versus Random Hypothesis* (HDRH) was recently posed by Chlamtáč et al. [CDK12, CDM17]. . Informally, the hypothesis postulates that no algorithm can distinguish (with high probability) a *random* hypergraph from an *adversarially chosen* hypergraph containing a dense subhypergraph. We defer its formal definition to Section 2.2.2. It is unknown whether HDRH implies ETH, yet HDRH seems to yield stronger lower bounds for problem such as Densest k-Subgraph. In the Densest k-Subgraph problem, the goal is to find a k-vertex subgraph of a graph with maximum number of edges. The problem does not admit an $\mathcal{O}\big(n^{1/(\log\log n)^c}\big)$-approximation algorithm under ETH as shown recently by Manurangsi [Man17], where $c > 0$ is a universal

constant. Under a simpler version of HDRH, a lower bound of $\Omega\big(n^{1/4-\varepsilon}\big)$ for any $\varepsilon > 0$ was shown by Chlamtáč et al. [CDK12].

1.1.2. Related work

Computing balanced cuts

Saran and Vazirani initiated the search for approximation algorithms for Minimum Bisection by showing an $n/2$-approximation [SV95]. Using recursive min-ratio cuts of the graph, Feige et al. improved the approximation factor to $\mathcal{O}(\sqrt{n}\log n)$ [FKN00]. Shortly after, Feige and Krauthgamer achieved the first polylogarithmic approximation ratio for Minimum Bisection in a breakthrough result [FK02]. Their algorithm's solution is $\mathcal{O}(\alpha \cdot \log n)$-approximate, where α is the approximation ratio of an algorithm for finding a min-ratio cut; plugging in the algorithm by Arora et al. gives $\alpha = \mathcal{O}(\sqrt{\log n})$ [ARV09]. Feige and Krauthgamer's algorithm constructs a decomposition tree of the graph using a variant of min-ratio cuts. The vertices of the decomposition tree are then combined via a nontrivial dynamic programming scheme. While this tree is reminiscent of the trees used as cut sparsifiers discussed below, their method does not generalize as far as cut sparsifiers do. In particular, the tree by Feige and Krauthgamer does not offer an immediate correspondence between cuts in the graph and cuts in the tree. Räcke later gave an algorithm for constructing a family of trees with $\mathcal{O}(\log n)$ average distortion of cut sizes [Räc08]. Minimum Bisection and other graph partitioning problems can be solved exactly on trees and allow the application of Räcke's framework, e.g., Multicut and Minimum k-multicut. The result immediately implies $\mathcal{O}(\log n)$-approximation algorithms for these problems.

Despite being a topic of active research for many years, there are few results concerning the approximability of Minimum Bisection. It is unknown whether $P \neq NP$ implies that no polynomial-time approximation scheme can exist for the problem. A polynomial-time approximation scheme (PTAS) is a family of $(1+\varepsilon)$-approximation algorithms for any $\varepsilon > 0$, Khot rules out the existence of a PTAS given the assumption that 3SAT cannot be solved by randomized subexponential time algorithms [Kho06]. Feige showed, assuming random 3SAT formulae under a natural distribution cannot be refuted in polynomial time, that there is no $(4/3-\varepsilon)$-approximation algorithm for any $\varepsilon > 0$ [Fei02].

To the best of our knowledge, no approximation algorithms for finding a minimum bisection in a hypergraph have been published prior to our work. This stands in contrast to the many results and software packages available for heuristic (hyper-)graph partitioning. We do not attempt to survey them here and instead refer to the work of Schultz and

Sanders [SS13], the survey by Buluç et al. [BMS+16] and references therein.

If the balance constraint is relaxed, the aforementioned results on approximating the min-ratio cut are key findings. The work of Leighton and Rao is based on rounding a linear programming relaxation [LR99] of the problem. Arora et al. gave a novel rounding algorithm for a semidefinite programming relaxation of the problem [ARV09]. Min-ratio cuts can be used to find *bicriteria* approximations to the minimum bisection. A (b, b')-bicriteria approximation algorithm for $b' < b \leq 1/2$ finds a a set of $b' \cdot n$ vertices whose cut cost is at most the cut cost of an *optimal* set of $b \cdot n$ vertices. Goemans et al. show how to obtain a $(1/2, 1/2 - \varepsilon)$ bicriteria approximation to Minimum Bisection with approximation factor α (reported in [LR99]). Here, α denotes the approximation factor of the min-ratio cut algorithm. For the task of partitioning a graph into k parts of equal size, a bicriteria approximation algorithm cannot be avoided as shown by Andreev and Räcke [AR06]. They observe that any approximation algorithm for the problem must decide when given a graph of disjoint cliques whether a single edge must be cut. In doing so, the algorithm would solve instances of the NP-complete 3Partition problem. Krauthgamer et al. give an $\mathcal{O}(\sqrt{\log n \log k})$-approximation that violates the balance constraint by a factor of 2 [KNS09]. Feldmann and Foschini demonstrate an algorithm that violates the balance constraint by only a factor of $(1 + \varepsilon)$ and obtain an $\mathcal{O}(\log n)$-approximation based on the tree decomposition by Räcke [FF15].

Interestingly, some of the work with the non-strict balance constraint extends well to hypergraph cuts and vertex cuts. For min-ratio cuts, Agarwal et al. present an $\mathcal{O}(\sqrt{\log n})$-approximation algorithm for the problem in directed graphs, which implies an algorithm with the same guarantees for vertex cuts and hypergraph cuts [ACMM05]. Via reductions, this implies a $(1/2, 1/2 - \varepsilon)$ bicriteria approximation algorithm to hypergraph bisection for any constant $\varepsilon > 0$. On the other hand, there is some evidence that finding exact hypergraph cuts and finding exact vertex cuts is hard. Lee shows that k-Vertex Separator, the problem of finding a set of vertices whose removal leaves every connected component with at most k vertices, is at least as hard as Densest k-Subgraph, a problem commonly believed to be hard to approximate [Lee19].

Trees as cut sparsifiers

A tree that acts as a sparsifier for the minimum s-t cuts in a graph is the classical Gomory-Hu tree named after its authors [GH61]. Gomory and Hu gave a polynomial-time construction of an edge-cut tree that exactly represents all minimum s-t-cuts in the graph. Furthermore, their tree is an s-t flow sparsifier of quality 1, which means that for any pair of vertices s, t, the value of the s-t flow in the original graph is the same as in the tree.

Trees that represent *all* cuts in the graph have been studied in the context of oblivious routing. An oblivious routing scheme is a probability distribution over (u, v)-paths for each vertex pair (u, v). Given a demand for each pair of vertices, the oblivious routing scheme specifies a path for each demand by drawing from the probability distributions. Routing the demand creates load on the edges of the network. The goal is to minimize the expected congestion, defined as the maximal ratio of load to capacity over all edges. Hence, while the Gomory-Hu tree is a flow sparsifier for $s-t$-flows only, an oblivious routing scheme requires a flow sparsifier for *multicommodity flows* of good quality.

Räcke showed that there exists a tree and a randomized embedding of this tree into the graph that is a flow sparsifier of quality $\mathcal{O}(\log^3 n)$ [Räc02]. Later, Harrelson et al. gave a polynomial-time construction of a suitable tree with quality $\mathcal{O}(\alpha \log n \log \log n)$, where α is the min-cut max-flow gap for multicommodity flows [HHR03]. As shown by Leighton and Rao, α is $\Theta(\log n)$ for general graphs [LR99]. In planar graphs and other graphs excluding any fixed graph as a minor, α is $\mathcal{O}(1)$ as shown by Klein et al [KPR93]. While any flow sparsifier is a cut sparsifier of the same quality, better cut sparsifiers might exist: Räcke and Shah show that the tree by Harrelson et al. is in fact a cut sparsifier of quality $\mathcal{O}(\log^{1.5} n \log \log n)$ [RS14].

While the above results use a *single tree*, Räcke considered a probability distribution over trees and obtained a flow sparsifier (and cut sparsifier) of quality $\mathcal{O}(\log n)$ [Räc08]. This asymptotically matches the lower bound of $\Omega(\log n)$ that holds for both tree flow sparsifiers and tree cut sparsifiers. For most, but not all applications, one can use the probability distribution over trees to solve cut problems in arbitrary graphs. One simply solves a given cut problem on each tree, and maps the best cut on a tree back to the graph at a loss of only $\mathcal{O}(\log n)$ in the approximation factor. The above strategy works well if the target function of the application is linear in the set of edges cut, e.g., for Minimum Bisection and Min-Sum Graph Partitioning. For problems such as Min-Max k-Partitioning [RS16] and Simultaneous Source Location [AGG+09], the use of a single tree is necessary.

We are not aware of any tree cut sparsifiers for hypergraphs or vertex cuts prior to our work. Negative results are only known insofar that oblivious routing is known to be considerably harder in directed graphs. Ene et al. show an $\Omega(n)$ lower bound for oblivious routing in general directed networks [EMPS16]. As the tree cut sparsifiers for graphs also yield polylogarithmic oblivious routing schemes, this suggests that similar trees might not exist for directed graphs.

1.1.3. Our results

This work improves the understanding of Minimum Hypergraph Bisection and gives upper and lower bounds on the approximability of this important problem. On the positive side, we obtain an $\tilde{\mathcal{O}}(\sqrt{n})$-approximation algorithm for Minimum Hypergraph Bisection. The $\tilde{\mathcal{O}}$ notation hides polylogarithmic factors in n and we defer the precise bounds to Chapter 2. Moreover, we obtain improved guarantees if either all hyperedges are large or all hyperedges are small. If all hyperedges have size at least $\Omega(n^\alpha)$, we show an $\tilde{\mathcal{O}}(n^{1-\alpha})$-approximation algorithm. If all hyperedges have size at most $\Omega(n^\alpha)$, a simple reduction to graph bisection gives an $\tilde{\mathcal{O}}(n^\alpha)$-approximation algorithm.

The approximation guarantees we reach for Minimum Hypergraph Bisection might seem unsatisfactory at first sight, as polylogarithmic approximation guarantees for the graph case have been known for a long time. Nevertheless, we show that algorithms with strong approximation guarantees are unlikely to exist for hypergraphs. More precisely, we prove that a polynomial-time approximation algorithm for Minimum Hypergraph Bisection with approximation guarantee $n^{1/(\log\log n)^c}$ for some universal constant c would violate the *Exponential Time Hypothesis*. Here $\text{poly}(x)$ denotes an arbitrary constant-degree polynomial in x. The proof of this result relies on a reduction from the Densest k-Subgraph problem, for which a similar hardness was recently shown by Manurangsi [Man17]. Furthermore, we show that Minimum Hypergraph Bisection does not admit an approximation algorithm of guarantee $\mathcal{O}(n^{1/4-\varepsilon})$, provided that the *Hypergraph Dense vs Random Hypothesis* holds. Analogous results are given for Minimum Vertex Bisection.

Further evidence that balanced hypergraph cuts are hard to approximate comes from our study of trees as cut sparsifiers for hypergraphs. We show that a *single tree* must have quality $\Omega(\sqrt{n})$ in the case of hypergraph cuts and unweighted vertex cuts. We complement this result by constructing a tree of quality $\tilde{\mathcal{O}}(\sqrt{nd_{\text{avg}}})$ for vertex cuts. We show how the tree can be applied for constructing minimum vertex bisections in graphs. Observe that we do not provide any lower bounds for convex combinations of cut trees such as the ones used by Räcke [Räc08]. Even so, this result marks a sharp distinction between the quality of edge cut trees that can be achieved in graphs (where polylogarithmic quality is possible) and hypergraphs.

1.2. Generalized Reordering Buffer Management

In the Reordering Buffer Management problem (RBM), a sequence of items arrives one-by-one and must be processed by the algorithm. The items represent some indivisible

unit available for processing, e.g., cars in a manufacturing plant or TCP requests in a streaming application. In order to distinguish between different types of items, we say that each item has a *color* which summarizes its properties. On arrival, the item is placed in a *buffer* that can hold up to b items. When the buffer is full, the algorithm must choose an item from the buffer to process next. If it processes two consecutive items of different colors, this incurs a cost of 1; more complicated cost models will be discussed later. The goal is to process all items while minimizing the total cost.

The above model was introduced by Räcke et al. [RSW02]. This thesis studies the *online* case of the problem, in which the algorithm does not know any future items when making a decision. Räcke et al. show that simple algorithms perform poorly for this problem. As an example, consider the *Largest-Color-First* algorithm (LCF) that picks, if possible, an item of the same color as the previously processed one, and otherwise an item of the color that is most frequent in the buffer. Now, suppose the first $b-2$ items have colors $1, 2, \ldots, b-3, b-2$. After that, the sequence consists of two items of color $b-1$ followed by two items of color b, and these last four items are repeated many times. LCF will only ever process the items of color $b-1$ and b, as those will be the largest colors in the butter. An optimal strategy instead begins by processing the $b-2$ first items and then performs LCF on the remaining sequence. For a sufficiently large sequence, this leads to LCF having $\Omega(b)$ times the cost of the optimal strategy, i.e., the LCF is $\Omega(b)$-*competitive*. Similar lower bounds can be shown for First-In-First-Out and Least-Recently-Used strategies. Räcke et al. also present the Bounded Waste (BW) strategy as the first nontrivial algorithm for the problem and prove that it is $\mathcal{O}(\log^2 b)$-competitive [RSW02].

RBM is not only relevant due to its elegant description and the algorithmic discoveries it inspired, but also because of its many applications. We have already seen the example of a paint shop in a manufacturing plant and highlight two more applications now. Krokowski et al. consider a 3D rendering system that assembles multiple primitives to the screen [KRSW04]. In such a system, the graphics processor slows down if subsequent primitives have different attributes such as texture or shader program. The authors show that using a reordering buffer can improve the performance of the system and they discuss several fast algorithms for the task. Azar et al. provided the example of a network device that receives packets from an input stream and must forward them to specific destinations [AEGK14]. If the device has no open connection to some destination, setting up a new connection incurs significant delay. When the device is limited to k open connections, e.g., $k = 1$, it can improve its performance by first storing all packets in a buffer. The device serves the open connections as long as packets for theses destinations exist. When none of the packets in the buffer can be distributed, the device must close a

connection and open a new one. Other applications have been identified in the context of information retrieval [BB02], storage systems [RSW02], and web caching [FMP+04].

A natural generalization of the problem is to consider multiple service stations available to process items. Räcke et al. [RSW02] name this problem *multi service sorting buffer*, yet the name Generalized Reordering Buffer Management (GRBM) has become more common. Even for a buffer of size one, this problem is nontrivial: The algorithm must decide for each item, which of its k service stations should (possibly) change its current color to process it. This is equivalent to the well-known Paging problem, that has been analyzed since the 80's; see, e.g., Sleator and Tarjan [ST85] and further discussion in Section 1.2.2. GRBM finds its usage, e.g., in the paint shop application (if there are multiple paint stations available) and the network device application discussed above. Azar et al. were the first to show algorithms for the online variant of this problem [AEGK14].

In this work, we are interested in algorithms for GRBM. We consider the uniform case, in which each color change costs exactly 1. Note that our algorithms also apply to classical RBM by setting the number of servers to one.

1.2.1. Problem statement

We now precisely define the model used for our discussion of GRBM. In this problem, the algorithm controls k servers located in a uniform n-point space, i.e., all points are at distance 1 from each other. Each point in the metric space is associated with a *color*. At each time step, an item of some color c arrives and is immediately placed in a *buffer*. We say that a buffer has size b, if it can hold up to $b+1$ items. If the buffer is full, the algorithm must remove at least one item from it before moving to the next time step, i.e., a buffer of size b contains b items at the end of the time step. If one of the servers currently resides on color c, the item is immediately removed from the buffer at no cost. Otherwise, the algorithm must specify a *server movement* in which it moves one of the servers to a new color and removes all items of the color from the buffer. The goal is to process the entire sequence with a minimum number of server movements.

The cost of processing a sequence in the above description is simply the total distance travelled by the servers. In the uniform cost model we consider, this is equal to the number of sever movements. Other works have considered a more refined scenario, in which the cost of a server movement depends on both the old and the new color of the server. The colors are then represented as points in some n-point metric space.

Competitive analysis

We use the standard model of *competitive analysis* to determine the performance of an algorithm in an online scenario. Here, we only review the most important notions from the field of online algorithms. See the textbook by Borodin and El-Yaniv for an in-depth introduction into the field [BE98].

For a given problem, let $c_{\mathrm{ALG}}(\sigma)$ denote the cost incurred by algorithm ALG on input sequence σ. Similarly, let $c_{\mathrm{OPT}}(\sigma)$ denote the cost that an *optimal* algorithm with unbounded resources and full knowledge about the sequence σ achieves. For minimization problems such as RBM, algorithm ALG is said to be c-competitive if for any input sequence σ

$$c_{\mathrm{ALG}}(\sigma) \leq c \cdot c_{\mathrm{OPT}}(\sigma) + d \ ,$$

where d is a constant independent of σ. We call the smallest c so that ALG is c-competitive the *competitive ratio* of the algorithm.

Randomized online algorithms are often framed as a two-player game between the algorithm on the one side and an *adversary* on the other side. The adversary knows the algorithm and has unlimited computational power. His task is to generate an input sequence for the algorithm. Ben-David et al. prove that the power of the adversary against a randomized algorithm heavily depends on the knowledge it obtains about the algorithm's random choices [BBK+94]. They distinguish three types of adversaries:

- **Oblivious adversary**: The adversary has no knowledge about the random choices and must construct the item sequence in advance.
- **Adaptive online adversary**: The adversary knows all previous decisions by the algorithm, but not its random choices for the next round.
- **Adaptive offline adversary**: The adversary knows all previous decisions and all future outcomes of the random choices.

They show that, in fact, the adaptive offline adversary is so powerful that randomization cannot improve an algorithm's competitive ratio against it.

This thesis studies randomized algorithms for the GRBM problem against an oblivious adversary. We are interested in the *expected cost* of a randomized online algorithm, where the expectation is over the random choices given a fixed input sequence. We say that randomized algorithm ALG is c-competitive if for any input sequence σ

$$\mathbb{E}[c_{\mathrm{ALG}}(\sigma)] \leq c \cdot c_{\mathrm{OPT}}(\sigma) + d \ ,$$

where d is a constant independent of σ. Again, we call the smallest c so that ALG is c-competitive the *competitive ratio* of the algorithm. In other words, the best possible deterministic online algorithm has the same competitive ratio as the best possible randomized online algorithm against an adaptive offline adversary.

Observe that the study of competitive analysis makes no assumptions on the running time of the online algorithm under investigation. As both RBM and GRBM have been shown to be NP-hard problems, it is unlikely that even a clairvoyant algorithm can find an optimal solution in polynomial time.

1.2.2. Related work

Generalized reordering buffer management.

Räcke et al. introduced GRBM alongside classical RBM under the name *multi service sorting buffer* [RSW02]. The authors observe that for a buffer of size one, it is equivalent to the paging problem. Chan et al. [CMSvS12] formally analyze the offline variant and show that it is NP-hard for any number of servers k by reducing RBM to it and showing that RBM is NP-hard. On the positive side, Azar et al. [AEGK14] present a deterministic online algorithm of competitive ratio $\mathcal{O}(\min\{k^2 \ln b, kb\})$ and a randomized online algorithm that attains competitive ratio $\mathcal{O}(\sqrt{b} \ln k)$. Their randomized algorithm is a variant of the *Marking* algorithm for the PAGING problem. Marking a color is delayed until $\sqrt{b}$ items of a color have arrived, while several clean-up procedures prevent a buffer overflow.

Reordering buffer management.

Both the online and offline variant of RBM have widely been studied. In this survey, we focus on results for the online variant of the problem; see [AR15] for further references. The first nontrivial algorithm for RBM was the *Bounded Waste* (BW) strategy by Räcke et al. [RSW02]. This algorithm is $\mathcal{O}(\log^2 b)$-competitive in the uniform cost model. Englert and Westermann developed the *Maximum Adjusted Penalty* algorithm (MAP) that is $\mathcal{O}(\log b)$-competitive [EW05]. This result even holds in a non-uniform setting where the cost of a color change depends on the new color. They also show that BW performs poorly in this case. In fact, Englert and Westermann prove that MAP is 4-competitive against an optimal algorithm with buffer size $b/4$ and that increasing the buffer by a constant factor only increases the cost by $\mathcal{O}(\log b)$. Avigdor-Elgrabli and Rabani developed the simple, deterministic *Threshold or Lowest Cost* (TLC) strategy with competitive ratio $\mathcal{O}(\log b / \log\log b)$ [AR15]. Their algorithm is combinatorial, yet they provide an LP-based

analysis. Finally, Adamaszek et al. presented a deterministic algorithm with competitive ratio $\mathcal{O}(\sqrt{\log b})$, nearly matching the lower bound of $\Omega(\sqrt{\log b / \log\log b})$ for deterministic algorithms shown in the same publication [ACER11].

Adamaszek et al. introduced the online block-devices problem, in which the server must return to a neutral location at the end of each time step [ACER12]. The authors gave a randomized online primal-dual algorithm of competitive ratio in $\mathcal{O}(\log\log b)$. For RBM in the uniform cost model, Avigdor-Elgrabli and Rabani gave a randomized $\mathcal{O}(\log\log b)$-competitive algorithm [AR13], asymptotically matching the lower bound given by Adamaszek et al. [ACER11]. Their algorithm first computes a nearly optimal, fractional solution to an LP relaxation of the problem. Second, a randomized rounding procedure transforms that fractional solution into a probability distribution over integral solutions. In contrast to the block-devices problem however, the authors do not use a packing-covering LP, hence the standard primal-dual framework does not apply. Instead, the authors combine a primal-dual analysis with a more complicated dual fitting argument.

For the non-uniform case, Avigdor-Elgrabli et al. achieved an exponential improvement over previous work by giving a randomized $\mathcal{O}((\log\log(b\gamma))^2)$-competitive algorithm, where γ is the ratio between shortest and longest distance [AIMR15].

In an even more general scenario, the switching cost between colors depends on both the old and the new color. In this scenario, the colors are arranged in a metric space. If the metric is an edge-weighted tree, algorithm PAY introduced by Englert et al. applies [ERW10]. PAY works in two steps: In the selection step, the items in the buffer can "buy" edges towards the position of the current color. As soon as this color can be reached via the "paid" edges, the algorithm starts the processing step. During the processing step, the algorithm visits all colors in the connected component of "paid" edges. Englert et al. show that PAY is $\mathcal{O}(D \log b)$-competitive on trees and $\mathcal{O}(\log^2 k \log n)$-competitive in general metric spaces. Here, the factor D denotes the hop-diameter of the tree, which is the number of edges on the longest path between two leaves. An improved analysis of PAY by Englert and Räcke shows that the algorithm is $\mathcal{O}(\log b)$-competitive on hierarchically separated trees (HSTs) [ER17]. An HST is a rooted, edge-weighted tree, whose edge weights increase by some constant with the distance to the root. Fakcharoenphol et al. show that any metric can be embedded into an HST so that the average stretch of an edge is $\mathcal{O}(\log n)$ [FRT04]. This implies that PAY has competitive ratio $\mathcal{O}(\log n \log b)$ in arbitrary metric spaces. Kohler and Räcke use a different technique based on finding *stable blocks* in the buffer and processing them [KR17]. This algorithm does not make use of HSTs and the authors show a competitive ratio of $\mathcal{O}(\log \Delta + \min\{\log n, \log b\})$, where Δ is the *aspect ratio* of the metric, the ratio of the longest to the shortest distance

between points in the metric. Their algorithm is *memory-robust*, i.e., its performance degrades gracefully with the size of the buffer.

Interestingly, both FRT-based algorithms and memory-robust algorithms can be shown to have competitive ratio $\Omega(\log n)$ on arbitrary metrics. The FRT-embedding loses a factor of $\mathcal{O}(\log n)$ and is optimal for general metric spaces. Bienkowski et al. show that any memory-robust algorithm must be $\Omega(\min\{b, \log n\})$-competitive [BBJ+18].

In another interesting scenario, colors are equidistant on an n-point line. Gamzu and Segev gave an $\mathcal{O}(\log n)$-competitive algorithm for the problem [GS09]. An elegant proof of this competitive ratio was given by Englert, who also notes that it is an interesting open problem whether a better competitive ratio is possible [Eng18]. He observes that the techniques employed so far are known to only give $\Omega(\log n)$-competitive results, so any improvement would require a fundamentally different approach (see also [BBJ+18]).

1.2.3. Delay problems

RBM falls into a broad class of reordering and aggregation problems that have recently gained much attention. Very broadly, these problems give a set of items, usually in a metric space, that need to be served by one or multiple servers. The *service cost*, which is the total distance travelled by the servers, may be reduced by rearranging and aggregating items. The algorithm's power to do so is, however, limited by charging a cost for delaying items. For example, RBM limits the number of unprocessed items via the buffer size. Next, we briefly survey other problems in this class.

In the Online Service with Delays (OSD) problem introduced by Azar et al., items arrive online in an n-point metric and can be served at any time after their arrival [AGGP17]. As for RBM, we associate the points of the metric space with colors. The cost of serving the sequence is the total delay of the items. In fact, each item may have its own (monotone, increasing) delay penalty function. The authors also introduce the k-OSD problem, in which there exist k servers to process the items.

While (G)RBM and (k-)OSD are similar in flavor, there are interesting technical differences between the problems. For RBM, no constant competitive online algorithm for uniform metrics can exist. For OSD, on the other hand, there is a simple 2-competitive strategy: Move the server to a color c whenever the items at c have accumulated delay penalty 1. One can see that between two movements of the online algorithm to c, the optimal algorithm must either have moved to color c as well, or accumulated delay penalty 1. In fact, Azar et al. show that k-OSD on uniform metrics can be reduced to classical Paging up to a constant factor.

Azar et al. also gave an $\mathcal{O}(kh^4)$-competitive algorithm for k-OSD on HSTs of height h.

Using a randomized FRT-embedding, this implies an $\mathcal{O}(k \log^5 n)$-competitive algorithm in general metrics. The authors note that their algorithm is quite different from the PAY algorithm used by Englert et al. for RBM on HSTs [ERW10]; in fact, a PAY-style algorithm performs poorly for delay problems. Very recently, Azar and Touitou gave an improved randomized algorithm for OSD with competitive ratio $\mathcal{O}(\log^2 n)$ on general metrics for $k = 1$ [AT19].

Online multilevel aggregation can be considered a variant of OSD. In this problem, items arrive on the leaves of a tree of depth D and each item comes with a penalty function. The cost of processing an item is the total weight of the path from the root to the item. An algorithm may process a set of items together, only paying the cost of the spanning tree connecting the items of the set with the root. Again, the cost of serving the sequence is the sum of service costs (weight of the spanning trees) and delay cost (total penalty of the items). The online multilevel aggregation problem was introduced by Khanna et al. [KNR02] and later studied by Bieńkowski et al. [BBB+16] who gave an $\mathcal{O}(D^4 2^D)$-competitive algorithm for the problem. Azar and Touitou gave a deterministic $\mathcal{O}(D^2)$-competitive algorithm for this problem. For the special case where all delay penalty functions are deadline functions (i.e., they are 0 before the deadline and infinite after it) Buchbinder et al. gave a deterministic algorithm of competitive ratio $\mathcal{O}(D)$ [BFNT17]. Interestingly, there remains a large gap to the best-known lower bound of 2 shown by Bieńkowski et al. [BBC+14].

Other than the examples mentioned above, there exist numerous results for classical online problems in a scenario with delays, e.g., online facility location [AT19], min-cost matching [EKW16], bin packing [AEvSV19], and set cover [CPSV18].

k-Server

The Paging problem, which we identified above as a special case of GRBM, arises naturally in computer systems using both slow, but large storage and a fast, but limited cache. Suppose a sequence of pages is requested online. At each time step, one of the n pages is requested and the online algorithm must move it into a cache that can hold up to k pages if the page is not already contained in it. Such a *page fault* incurs unit cost, and the algorithm's goal is to minimize the total cost. The central decision by the paging algorithm lies in the question which page to evict upon a page fault. Sleator and Tarjan gave the first online analysis of the *Least-Recently-Used* algorithm (LRU) and showed that it is k-competitive [ST85]. This strategy always evicts the page from the cache that has been requested least recently. Manasse et al. showed that no deterministic algorithm can have competitive ratio below k, making LRU an optimal strategy for

paging [MMS90]. For randomized algorithms, that lower bound is the k-th harmonic number $H_k = \Theta(\log k)$, which is nearly matched by the $(2H_k - 1)$-competitive *Marking* algorithm developed by Fiat et al. [FKL+91]. Achlioptas et al. provide a tight analysis of algorithm *Marking* [ACN00]. Multiple H_k-competitive algorithms are known, the earliest was given by McGeoch and Sleator [MS91].

While today optimal algorithms are known for the paging problem, the same is not true for its generalization, the k-Server problem. In this problem introduced by Manasse et al., k servers must serve items in a metric space online [MMS90]. The paging problem corresponds to the uniform metric in which all points are at unit distance from each other. The k-server problem has famously been conjectured to admit a deterministic k-competitive algorithm for any value of k, *the k-server conjecture* [MMS90]. The conjecture seemed far out of reach at the time it was posed, as not even an algorithm with competitive ratio depending only on the number of servers k was known. Fiat et al. gave the first such algorithm, albeit with an exponential dependency on k [FRR94]. Koutsoupias and Papadimitriou then showed that their *work function* algorithm has competitive ratio $2k - 1$ [KP95]. The work function algorithm has been the best-known deterministic k-server algorithm for more than 25 years now. For a modern look on the algorithm and a brief survey of the problem, see the paper by Koutsoupias [Kou09].

The power of randomization for the k-Server problem has recently received strong attention. The *randomized k-server conjecture* states that a randomized $\mathcal{O}(\log k)$-competitive algorithm for the problem exists. The conjecture implies that k-Server admits the same approximation ratio in arbitrary metric spaces as in a uniform metric space. Significant progress towards the resolution of this conjecture was made by Bansal et al., who showed an $\mathcal{O}(\log k)$-competitive algorithm for k-Server in star metrics [BBN12]. Their algorithm applies the primal-dual scheme developed by Buchbinder and Naor [BN09]. With a more complicated linear programming relaxation, they extended their techniques to develop an $\mathcal{O}(\log^2 k \log^3 n \log \log n)$-competitive algorithm for the general case [BBMN15]. Although this algorithm's performance still depends on n, it was the first algorithm to only have polylogarithmic dependency in both n and k. Bubeck et al. used a novel linear programming relaxation and techniques borrowed from online learning algorithms to develop an $\mathcal{O}(\log^2 k \log n)$-competitive algorithm for the k-server problem [BCL+18]. Their result is a central building block in a recent breakthrough by Lee who gave an algorithm of competitive ratio $\mathcal{O}(\log^5 k)$.

1.2.4. Our results

The main result of Chapter 3 is a randomized $\mathcal{O}(\log k(\log k + \log\log b))$-competitive algorithm for GRBM on uniform metric spaces. This is the first algorithm to achieve polylogarithmic competitiveness for the problem. For constant k, our algorithm asymptotically matches the lower bound of $\Omega(\log\log b + \log k)$. This lower bound is on the one hand inherited from the connection of GRBM to RBM (with the case of a single server), which has a lower bound of $\Omega(\log\log b)$. On the other hand, GRBM turns into the paging problem for $b = 1$, which gives a lower bound of $\Omega(\log k)$.

Our algorithm also implies an $\mathcal{O}(\log\log b)$-competitive algorithm for classical RBM. This reproves a result by Avigdor-Elgrabli and Rabani [AR13]. Interestingly, our algorithm uses a different framework and is based on a novel linear programming relaxation for GRBM. This new linear program builds upon the work by Adamaszek et al. for scheduling block devices [ACER12].

1.3. Bibliographical Notes

The results of Chapter 2 have been obtained in joint work with Roy Schwartz and Harald Räcke. They have been presented in preliminary form at the 30th ACM Symposium on Parallelism in Algorithms and Architectures [RSS18].

The proofs shown in this chapter are mostly extended and revised version of those shown in [RSS18]. In addition to providing full proofs for all results, this thesis also shows how to adapt the Minimum Hypergraph Bisection procedure to Minimum Vertex Bisection. It also shows how a vertex cut tree can be used to find a minimum vertex bisection.

The results of Chapter 3 have been obtained in joint work with Matthias Englert and Harald Räcke and have been presented in preliminary form at the 60th Annual IEEE Symposium on Foundations of Computer Science [ERS19]. The proofs shown in this chapter are mostly extended and revised versions of those shown in [ERS19]. Section 3.4 gives an entirely new competitiveness proof based on a more rigorous analysis of the probability space.

2. Balanced Cuts in Graphs and Hypergraphs

This chapter is devoted to our results on computing balanced partitions in graphs and hypergraphs. We begin by analyzing more precisely the relationship between edge cuts in graphs, hyperedge cuts in hypergraphs and vertex cuts in graphs and we explain the folklore reduction between them. After this warm-up exercise, we turn to Minimum Hypergraph Bisection and we give an approximation algorithm for this problem. We also analyze the computational hardness of finding a good hypergraph bisection by proving lower bounds based on the complexity hypotheses HDRH and ETH. In the third part of this chapter, we first give a construction for a vertex cut tree approximating hypergraph cut and vertex cuts. We then give four lower bounds that demonstrate the limits in constructing such trees.

2.1. Preliminaries

This work essentially considers three types of cut problems. The first type are classical *edge cuts in undirected graphs*. In this scenario, an undirected graph $G = (V, E)$ is disconnected by removing edges from the edge set E. The second scenario is a natural extension, namely *hyperedge cuts in hypergraphs*. Here, a hypergraph $H = (V, E_H)$ is disconnected by removing hyperedges from the set of hyperedges E_H. The third scenario is *vertex cuts in undirected graphs* where an undirected graph is disconnected by removing vertices from the vertex set.

This initial section demonstrates reductions between the three cut problems we consider. The reductions are mostly simple and well-known in the literature. We include them for completeness and as a reference for future sections.

Edge Cuts in Graphs and Hyperedge Cuts. We begin by exploring the relationship between edge cuts in undirected graphs and hyperedge cuts in hypergraphs. Every undirected graph is clearly a 2-uniform hypergraph, so any algorithm for cutting hypergraphs works on graphs as well. On the other hand, the following construction transforms a

hypergraph into a weighted, undirected graph so that any cut size is distorted by at most the maximum hyperedge size.

Given a hyperedge $H = (V, E_H)$, replace each hyperedge $h \in E_H$ with a clique C_h among the vertices of h of edge weights $1/(|h| - 1)$. Call the resulting multigraph (with parallel edges) G'. In order to obtain a graph G out of the multigraph G', replace parallel edges with a simple weighted edge.

Proposition 2.1. *For any hypergraph $H = (V, E_H)$ of maximum hyperedge size $h_{\max}$, there is a weighted graph $G = (V, E_G)$ over the same vertex set so that for any $A \subset V$,*

$$\delta_G(A) \leq \delta_H(A) \leq \min\{|A|, h_{\max}/2\} \cdot \delta_G(A) \ .$$

Graph G can be constructed in polynomial time and $|E_H| \leq |E_G| \leq h_{\max}^2 \cdot |E_H|$.

Proof. Given hypergraph H, use the above construction to obtain the undirected, weighted graph G. Suppose hyperedge h is cut by a set $A \subset V$ and let $x = |h \cap A|$. It follows that the number of edges of clique C_h that is cut by A is

$$|h \cap A| \cdot |h \cap (V \setminus A)| = x(|h| - x) =: f(x) \ .$$

On the one hand, function f is concave in x, so it is minimized for $x = 1$ or $x = |h| - 1$. For both cases, this gives that at least $|h| - 1$ edges of C_h are cut. As the edge weight in C_h is $1/(|h| - 1)$, the lower bound follows.

On the other hand, the number of edges in the cut is at most $|A|(|h| - 1)$, which gives the upper bound of $|A|$. Furthermore, function f is maximized at $x = |h|/2$, so $f(x) \leq |h|^2/4$. The weight of edges cut is therefore $|h|^2/(4|h| - 4) \leq |h|/2$ as $|h| \geq 2$, which gives the second upper bound. □

Vertex Cuts in Graphs and Hyperedge Cuts. We now demonstrate a reduction from hyperedge cuts to vertex cuts in undirected graph. The following construction is essentially given by Leighton and Rao [LR99].

Given a hypergraph $H = (V, E_H)$, we construct a bipartite graph $G = (V \cup E_H, E_G)$. Graph G is bipartite between V and E_H. For any $h \in E_H$ and $v \in h$, construct an edge $\{v, h\}$ to obtain the set E_G. For vertices in E_H, set the vertex weight to 1. For vertices $v \in V$, set the vertex weight to $\deg_H(v) + 1$, where $\deg_H(v)$ denotes the number of hyperedges in H vertex v is contained in.

Proposition 2.2 ([LR99]). *For any hypergraph $H = (V, E_H)$, there is a vertex-weighted bipartite graph $G = (V \cup E_H, E_G, w)$ so that for any disjoint $A, B \subset V$*

$$\delta_H(A, B) = \gamma_G(A, B) \ .$$

Graph G can be constructed in polynomial time and has total vertex weight $|E_H| + |V|(d_{\text{avg}}(H) + 1)$, where $d_{\text{avg}}(H)$ denotes the average vertex degree in H.

Proof. Given hypergraph H, use the above construction to obtain the undirected, vertex-weighted graph G. Observe that a minimum-weight vertex separator of any two vertex sets in G never picks a vertex of V. To see this assume for a contradiction that the minimum-weight separator S of A and B contains vertex $v \in V$. Replacing v by all of its neighbors (in G) still separates A and B, yet the weight of S decreases by 1, which contradicts the optimality of S. Therefore, optimal vertex separators in G are taken from E_H only and thus correspond to collections of hyperedges in H. As $w(h) = 1$ for all $h \in E_H$, the claim follows. □

Leighton and Rao actually give a variant of the above construction [LR99]. They use two weight functions w and b on the vertex set, where $w(v)$ is the *cut weight* of a vertex v and $b(v)$ is its *balancing weight*. Vertices of G in V obtain infinite cut weight and balancing weight 1. Vertices of G in E_H obtain cut weight 1 and balancing weight 0. This transformation allows a one-to-one correspondence between edge partitions in H and non-infinite cost vertex partitions in G. Let γ_G^w denote the cost of a vertex cut with respect to weight function w, then we obtain the following theorem.

Proposition 2.3 ([LR99]). *For any hypergraph $H = (V, E_H)$, there is a vertex-weighted bipartite graph $G = (V \cup E_H, E_G, w, b)$ with two vertex weight functions w, b so that for any disjoint $A, B \subset V$*

$$\delta_H(A, B) = \gamma_G^w(A, B) \ .$$

Graph G can be constructed in polynomial time and has total vertex weight $b(V \cup E_H) = |V|$.

2.2. Approximating Minimum Hypergraph Bisection

2.2.1. Approximation algorithms

The approximation algorithm for MHB shown in this section is an adaptation of the algorithm by Feige et al. for Graph Bisection to hypergraphs [FKN00]. We begin by sketching their algorithm.

Their algorithm proceeds in two phases. In the first phase, it partitions the graph into small subgraphs using recursive min-ratio cuts. The goal is that each of the subgraphs returned by the first phase is *nearly monochromatic*. This means that the subgraph lies almost entirely on one side of some fixed optimal bisection. In the second phase, the algorithm tries to compute, for each subgraph, the cut induced by the optimal bisection within the subgraph. This divides each subgraph in two. Finally, all subgraphs are grouped to form a bisection. The algorithm's approximation ratio in graphs is $\tilde{\mathcal{O}}(\sqrt{n})$. We show the same ratio is achieved in hypergraphs.

Before presenting the overall algorithm in detail, we consider the problem of cutting a few vertices from a hypergraph. This is used later as a subroutine in the second phase of the algorithm we present.

Cutting k vertices from a hypergraph

The problem of removing k vertices from a hypergraph is called the Unbalanced k-Cut problem. We give two simple algorithms approximating an optimal solution.

Proposition 2.4. *There exists a* $\min\{k, h_{\max}/2\} \cdot \mathcal{O}(\log n)$*-approximation algorithm for* Unbalanced k-Cut *in hypergraphs where* $h_{\max}$ *is the size of the largest hyperedge.*

Proof. Unbalanced k-Cut can be solved in graphs with approximation factor $\mathcal{O}(\log n)$ using the convex combination of edge cut trees by Räcke [Räc08]. The result therefore follows by applying the transformation of hypergraphs into graphs of Proposition 2.1. □

If all hyperedges span more than k vertices, any set of k vertices has no internal hyperedges. In this case, Unbalanced k-Cut asks to find k vertices so that the total number of hyperedges they are incident to is minimal. This is an instance of Minimum k-Union (MkU). This problem was studied recently by Chlamtáč et al [CDM17] and is discussed in detail in Section 2.2.2. Chlamtáč et al. obtain the following result.

Proposition 2.5 ([CDM17])**.** *If all hyperedges have at least size* k*, and* $k = \Omega\big(n^{1-\alpha}\big)$*, then there is an* $\tilde{\mathcal{O}}(n^{\alpha(1-\alpha)+\varepsilon})$*-approximation algorithm for the unbalanced* k*-cut problem, for any constant* $\varepsilon > 0$*.*

Approximation algorithm for minimum hypergraph bisection

We are now ready to adapt the algorithm by Feige et al. to hypergraphs [FKN00].

Suppose we are given a hypergraph $G = (V, E)$ on n vertices. The algorithm first guesses the value OPT of an optimum solution. This can be done by binary search on the

range of possible values, as this only increases the running time by a polynomial factor. We assume from here on that the value of OPT is known.

The first phase of the algorithm uses an approximation algorithm $\mathcal{A}$ for min-ratio cut in hypergraphs and a parameter k to be determined later. Algorithm $\mathcal{A}$ is then used recursively on hypergraph G as follows. We start with the entire graph G and have $\mathcal{A}$ compute an approximate min-ratio cut (V_1, V_2) of the vertex set of G. If the sparsity of that cut is more than $\alpha \operatorname{OPT}/k$, we stop. If, on the other hand, the sparsity is at most $\alpha \operatorname{OPT}/k$, let G' and G'' be the subhypergraphs induced by V_1 and V_2, respectively. On both subhypergraphs G' and G'', continue this process recursively. When the recursive process ends, the graph has been partitioned into subhypergraphs G_i. Each subhypergraph G_i has the property that algorithm $\mathcal{A}$ has not found a cut of sparsity less than $\alpha \operatorname{OPT}/k$ in G_i. It follows that no cut of sparsity less than OPT/k exists in G_i. In other words, the recursive procedure results in a partitioning of G into well-connected subhypergraphs. This concludes the first phase of the algorithm. The next two lemmas show the main properties of the hypergraphs G_i.

Lemma 2.6. *The total weight of edges cut in the first phase is at most*

$$\alpha \cdot n \log n \frac{\operatorname{OPT}}{k} \ .$$

Proof. Every cut S_i of the first phase fulfills the inequality $|\delta(S_i)|/|S_i| \leq \alpha \operatorname{OPT}/k$, where S_i denotes the smaller side of the cut. This gives $|\delta(S_i)| \leq \alpha |S_i| \operatorname{OPT}/k$.

We can amortize the increase $\delta(S_i)$ of the total cut-cost in the i-th step to the vertices on the smaller side. Since during the whole construction a vertex appears on the smaller side of a cut at most $\log n$ times, we amortize at most $\alpha \log n \cdot \operatorname{OPT}/k$ against a single vertex. Summing over all vertices gives the lemma. □

Let G_i denote the subhypergraphs returned after the first phase. We fix some optimal bisection represented as a black and white coloring of the vertices. Subhypergraph G_i has *minority color* white, if it contains fewer white vertices than black vertices; otherwise, its minority color is black. The vertices that have the minority color are called *minority vertices*. The following lemma shows that the first phase ends with only few minority vertices in the subhypergraphs.

Lemma 2.7. *There exist less than k minority vertices by the end of the first phase.*

Proof. Let k_i denote the number of minority vertices in part G_i and let OPT_i be the number of hyperedges of the (fixed) optimal bisection inside G_i. Note that $\sum_i \operatorname{OPT}_i \leq$

OPT, as the subhypergraphs G_i are disjoint. As no cut of sparsity below OPT/k exists, every i must fulfill

$$\frac{\mathrm{OPT}_i}{k_i} \geq \frac{\mathrm{OPT}}{k} \quad .$$

Summing over all i gives

$$\frac{\sum_i \mathrm{OPT}_i}{\mathrm{OPT}} \geq \frac{\sum_i k_i}{k} \quad .$$

As the left-hand side is at most 1, the same holds for the right-hand side. The total number of minority vertices $\sum_i k_i$ is therefore at most k. □

The second phase of the algorithm aims to approximate the optimum cut within each piece G_i. For this we "guess" in each subhypergraph the number of minority vertices k_i and remove this many vertices using the algorithm from Proposition 2.4. These guesses are made one-by-one within a dynamic program by Feige et al. which we describe next [FKN00].

Feige et al.'s dynamic programming algorithm labels each part G_i either black or white. It furthermore determines for each part the number k_i of minority vertices in the part. The dynamic programming table is $C_j(W, B, m_W, m_B)$. A table entry stores the (approximate) cost of labeling parts $G_1, \ldots, G_j$ so that parts of W vertices in total are labeled white and parts of B vertices in total are labeled black. The total number of minority vertices of white parts and black parts are m_W and m_B, respectively.

For $j = 1$, suppose that we label $G_1 = (V_1, E_1)$ white. Then $C_1(|V_1|, 0, \ell, 0)$ equals the cost of cutting $\ell \leq k$ vertices from G_1 using the algorithm from Proposition 2.4. When labeling G_1 black, the table entries follow by symmetry. For $j > 1$, we guess whether G_j is labeled black or white and then the number of minority vertices. There are only $2n$ for each part, which gives a polynomial running time of the dynamic program.

If the total number of parts is p, we choose the labeling of parts corresponding to the smallest table entry $C_j(W, B, m_W, m_B)$ with $W - m_W + m_B = n/2$ (so that we can combine the parts to a bisection) and $m_W + m_B \leq k$ (so that there are at most k minority vertices).

Lemma 2.8. *The total cost of the cut computed in the second phase is* $\mathcal{O}(\log n)k\,\mathrm{OPT}$.

Proof. Suppose the optimum solution has k_i minority vertices from subhypergraph G_i. This means that a k_i-unbalanced cut through G_i of cost at most OPT exists. Our algorithm therefore finds a cut of cost at most $k_i\,\mathrm{OPT}\,\mathcal{O}(\log n)$ in G_i. Summing over all subhypergraphs gives cost at most

$$\mathcal{O}(\log n) \sum\nolimits_i k_i\,\mathrm{OPT} \leq \mathcal{O}(\log n)k\,\mathrm{OPT} \quad .$$

We have used in the inequality that the total number of minority vertices is at most k by Lemma 2.7. □

Theorem 2.9 ([RSS18]). *There is a polynomial time $\mathcal{O}(\sqrt{n}\log^{5/4} n)$-approximation algorithm for* Minimum Hypergraph Bisection.

Proof. From lemmas 2.6 and 2.8 we know that the total cost is at most

$$\left(\frac{\alpha n}{k} + k\right) \mathrm{OPT} \log n \ .$$

The theorem follows by setting $k = \sqrt{\alpha n}$ and using $\alpha = \mathcal{O}(\sqrt{\log n})$. □

If either all hyperedges are large or all hyperedges are small we can obtain the following improved bounds.

Theorem 2.10 ([RSS18]). *If all hyperedges have size at least $\Omega(n^\alpha)$, there exists an $\tilde{\mathcal{O}}(n^{1-\alpha})$ approximation algorithm for* Minimum Hypergraph Bisection.

If all hyperedges have size at most $\mathcal{O}(n^\alpha)$, there is an $\tilde{\mathcal{O}}(n^\alpha)$-approximation algorithm for Minimum Hypergraph Bisection.

Proof. If all hyperedges have at least size $z = \Omega(n^\alpha)$, we choose $k = z$ for the first phase of the algorithm. Then the total number of minority vertices for the second phase is at most z by Lemma 2.7. This means we can use the algorithm from Proposition 2.5 for the second phase. The approximation factor of the overall algorithm becomes then $\tilde{\mathcal{O}}(n^{1-\alpha}) + \tilde{\mathcal{O}}(n^{\alpha(1-\alpha)+\varepsilon}) \in \tilde{\mathcal{O}}(n^{1-\alpha})$.

If all hyperedges have size *at most* $\mathcal{O}(n^\alpha)$, applying Proposition 2.1 and a logarithmic approximation algorithm for Minimum Bisection in simple graphs gives an $\tilde{\mathcal{O}}(n^\alpha)$ approximation algorithm for Minimum Hypergraph Bisection. □

2.2.2. Hardness results

This section shows that a polynomial-time algorithm with approximation ratio $n^{1/(\log\log n)^c}$ for MHB would violate the *Exponential Time Hypothesis* (ETH). The stronger *Hypergraph Dense versus Random Hypothesis* (HDRH) is shown to imply that the approximation ratio for MHB cannot be better than $n^{1/4-\varepsilon}$. Both results indicate a stark contrast between classical Minimum Bisection and its variant for hypergraphs MHB, as polylogarithmic approximation algorithms for the graph case have long been known, see Feige and Krauthgamer [FK02].

Hardness based on hypergraph dense versus random

The hardness results of this section stem from the connection between MHB and the Minimum k-Union (MkU) problem. In MkU, one is given a hypergraph $G = (V, E)$ of n vertices and m hyperedges. The goal is to pick $1 \leq k \leq m$ hyperedges so that their union has minimum cardinality. MkU has been introduced recently by Chlamtáč et al. who gave an $\tilde{\mathcal{O}}(m^{1/4+\varepsilon})$-approximation algorithm [CDM17]. The authors observe that MkU is closely related to Densest k-Subgraph (DkS), the problem of finding in a graph a subgraph on k vertices with maximum number of edges. We have seen in Section 2.2.1 that this problem is equivalent to the Unbalanced k-Cut problem if all hyperedges have size more than k. Our goal is now to analyze this connection more precisely. We start by defining some concepts from random hypergraphs.

For integers n, $r \leq n$, and a probability $0 < p < 1$, the *random graph* $\mathcal{G}_{n,p,r}$ is a graph on n vertices constructed via the following random process. Independently for any r-subset S of vertices, add a hyperedge S with probability p to the hypergraph. Writing $G \sim \mathcal{G}_{n,p,r}$ means that G is constructed through this random process.

We sometimes say that some property P holds *with high probability*. This means that the property P holds with probability at least $1 - \frac{1}{n^c}$ for an arbitrary constant c. Sometimes the property P will be parameterized by a different constant c' (hidden in the $\mathcal{O}$-notation) that may depend on c.

Chlamtáč et al. show that for any $\varepsilon > 0$ no $m^{1/4-\varepsilon}$-approximation algorithm for MkU exists if the HDRH holds.

This hypothesis is based on the Hypergraph Dense versus Random problem. This game is a variation of the Dense versus Random distinguishing (promise) problem introduced (implicitly) by Bhaskara et al. [BCC+10]. Informally, Dense versus Random is a simple two-player game. At the beginning of each round, a random graph is generated based on the game's parameters. Player 1 can either use the random graph or construct a graph containing some subgraph of high density. Player 2 must then guess if Player 1 presented the random graph or not. The goal of Player 2 is to guess correctly with high probability. An in-depth discussion of Dense versus Random, its variants and algorithms tackling it can be found in Vijayaraghavan's thesis [Vij12].

Hypergraph Dense versus Random adapts Dense versus Random for hypergraphs. Formally, the problem is parameterized by an integer k and constants r and $0 < \alpha, \beta < r - 1$. The problem asks to distinguish between the following two graphs:

Dense G is adversarially chosen so that a planted subhypergraph of G on k vertices has log-density β,

Random $G \sim \mathcal{G}_{n,p,r}$, where $p = n^{\alpha-r+1}$.

The log-density of a hypergraph on k vertices and average vertex degree d_{avg} is $\log_k(d_{\text{avg}})$. It follows that the average degree within the subhypergraph of the dense case is k^β.

Hypothesis 2.11. *[HDRH, [CDM17]] For all constant r and $0 < \alpha, \beta < r - 1$, for all sufficiently small $\varepsilon > 0$, and for all k such that $k^{1+\beta} \leq n^{(1+\alpha)/2}$, we cannot solve* Hypergraph Dense vs Random *with log-density α and planted log-density β in polynomial time (with high probability) when $\beta < \alpha - \varepsilon$.*

We say that an instance of MkU is *quasi α-regular* if the hypergraph is r-uniform for some r and the degree of each vertex is $\Theta(n^\alpha)$. A quasi α-regular instance of MkU is written as (G, n, m, α, r). We first show how the lower bound on the approximation ratio for quasi α-regular MkU-instances predicted by HDRH is parameterized by α. In doing so, we extend the result by Chlamtáč et al. [CDM17]. This will later serve us for obtaining precise lower bounds for uniform MHB instances.

Lemma 2.12. *Assuming Hypothesis 2.11 (HDRH), there is no polynomial-time algorithm for* MkU *that achieves approximation factor $\mathcal{O}(m^{\min\{\alpha/(2\alpha-2),\alpha/(1+\alpha)^2\}-\varepsilon})$ with high probability for sufficiently small ε.*

Proof. In order to establish the lemma we first show some facts about $\mathcal{G}_{n,p,r}$. The proofs of these facts have been deferred to Appendix A.1.

Claim 2.13. *A random graph $\mathcal{G}_{n,p,r}$ fulfills the following properties.*

1. *Any vertex of a $\mathcal{G}_{n,p,r}$ with $p = n^{1+\alpha-r}$ has vertex degree $\Theta(n^\alpha)$, with high probability.*
2. *Any set of n hyperedges in a $\mathcal{G}_{n,p,r}$ with $p = n^{1+\alpha-r}$ covers at least $\Omega\big((n/p)^{1/r}\big)$ vertices, with high probability.*
3. *Any set of $n^{(1+\alpha)/2}/r$ hyperedges in a $\mathcal{G}_{n,p,r}$ with $p = n^{1+\alpha-r}$ covers at least $\Omega\big(n^{(1+\alpha)/2-\varepsilon}\big)$ vertices, with high probability, if $\alpha < 1$, r is sufficiently large and ε is a small constant.*

We show that an approximation algorithm for MkU distinguishes between the *Dense* case and the *Random* case of the Hypergraph Dense vs Random problem. We use the hypergraph provided by Hypergraph Dense vs Random as input for MkU. By Fact 1, this is a quasi α-regular instance of MkU in the *Random* case with high probability. In the *Dense* case, the instance chosen by the adversary must therefore also be quasi α-regular, otherwise distinguishing them would be trivial.

In the *Dense* case, the adversary has planted a subhypergraph of k vertices and log-density β in the graph. This subhypergraph contains $\ell = k^{1+\beta}/r$ edges as k^β is the average vertex degree and r the size of the hyperedges. It follows that in the *Dense* case, the minimum union of ℓ hyperedges has size at most k.

Let $\hat{k}$ be the size of the minimum ℓ-union in the *Random* case, i.e., the minimum size of a union of ℓ hyperedges in a $\mathcal{G}_{n,p,r}$. We show that $\hat{k}$ is considerably larger than k with high probability. Assuming Hypothesis 2.11, no polynomial time algorithm may distinguish between k and $\hat{k}$. This gives a lower bound of $\hat{k}/k$ on the approximation ratio for algorithms approximating MkU.

In the remainder of the proof, we show how to set the parameters of the Hypergraph Dense versus Random instance so as to obtain the desired gap $\hat{k}/k$. Recall that we aim for a result that is parameterized by α. We distinguish two cases depending on the value of α.

$\alpha > 1$. We choose $\ell = n$ (note that this in fact chooses k for the Dense vs Random instance, because we have set $\ell = k^{1+\beta}/r$ before). We first show that $k^{1+\beta} \leq n^{(1+\alpha)/2}$ as this is a constraint on k that must be fulfilled for Hypothesis 2.11. Indeed, $k^{1+\beta} = r\ell = rn < n^{(1+\alpha)/2}$ for $\alpha > 1$ and n sufficiently large.

Fact 2 implies that with high probability $\hat{k}$ will be at least $\Omega\big((n/p)^{1/r}\big)$. Let m denote the total number of edges in the random instance. Fact 1 implies that, with high probability, $m = \Theta(n^{1+\alpha})$. We use this fact and rewrite $(n/p)^{1/r}$ with respect to m:

$$\begin{aligned}(n/p)^{1/r} &= \Theta\left(\left(m^{1/(1+\alpha)}n^{r-(1+\alpha)}\right)^{1/r}\right)\\ &= \Theta\left(\left(m^{1/(1+\alpha)}m^{r/(1+\alpha)-1}\right)^{1/r}\right)\\ &= \Theta\left(m^{1/(r(1+\alpha))+1/(1+\alpha)-1/r}\right)\ .\end{aligned}$$

In the first equality, we used that $p = n^{1+\alpha-r}$. As $k = (rn)^{1/(1+\beta)}$ which is in $\Theta(m^{1/((1+\alpha)(1+\beta))})$ (with r being constant and $\beta > 0$), it follows that

$$\hat{k}/k = \Omega\left(m^{\frac{1}{r(1+\alpha)}-\frac{1}{r}+\frac{1}{1+\alpha}-\frac{1}{(1+\alpha)(1+\beta)}}\right)\ .$$

We now choose $\beta = \alpha - \varepsilon$ and $\varepsilon/2 > 1/r$, then the exponent is at least

$$\begin{aligned}\frac{1}{1+\alpha} - \frac{1}{(1+\alpha)(1+\alpha-\varepsilon)} - \frac{1}{r} &= \frac{1}{1+\alpha}\left(1 - \frac{1}{1+\alpha-\varepsilon}\right) - \frac{1}{r} \\ &\geq \frac{\alpha}{(1+\alpha)^2} - \frac{\varepsilon}{(1+\alpha)^2} - \frac{\varepsilon}{2} \\ &> \frac{\alpha}{(1+\alpha)^2} - \varepsilon \ .\end{aligned}$$

The second inequality uses that $1 + \alpha > 2$. It follows that the gap $\hat{k}/k$ is at least $\Omega\big(m^{\alpha/(1+\alpha)^2-\varepsilon}\big)$.

$\alpha \leq 1$. We choose $\ell = n^{(1+\alpha)/2}/r$. This choice is valid as $k^{1+\beta} = \ell r = n^{(1+\alpha)/2}$. Fact 3 implies that with high probability $\hat{k}$ will be at least $\Omega(n^{(1+\alpha)/2-\varepsilon})$ if r is sufficiently large. Again the total number of edges in a random hypergraph is $m = \Theta(n^{1+\alpha})$ by Fact 1, so

$$\hat{k} = \Omega(m^{1/2-\varepsilon/(1+\alpha)}) \ .$$

On the other hand, $k = n^{(1+\alpha)/(2(1+\beta))} = \Theta(m^{1/(2(1+\beta))})$, so the gap $\hat{k}/k$ is

$$\hat{k}/k > \Omega\big(m^{1/2-\varepsilon/(1+\alpha)-1/(2(1+\beta))}\big) \ .$$

Using $\beta = \alpha - \varepsilon$ and $\varepsilon' = 3\varepsilon/2$, the exponent is

$$\begin{aligned}\frac{1}{2} - \frac{1}{2(1+\beta)} - \frac{\varepsilon}{1+\alpha} &= \frac{1}{2} \cdot \frac{\alpha-\varepsilon}{1+\alpha-\varepsilon} - \frac{\varepsilon}{1+\alpha} \\ &> \frac{1}{2} \cdot \frac{\alpha-\varepsilon}{1+\alpha} - \frac{\varepsilon}{1+\alpha} \\ &> \frac{\alpha}{2(1+\alpha)} - \frac{3\varepsilon}{2(1+\alpha)} > \frac{\alpha}{2(1+\alpha)} - \varepsilon'\end{aligned}$$

It follows that the gap $\hat{k}/k$ is at least $\Omega\big(m^{\alpha/(2+2\alpha)-\varepsilon'}\big)$.

The combination of both cases directly yields the lemma. □

An instance of MkU can be transformed into a suitable instance of MHB using a construction we describe now. At a high level, our reduction takes the instance G of MkU and first constructs the dual hypergraph G^* by switching vertices and hyperedges. Then a new supervertex that is incident to every hyperedge of G^* is added to G^*. The k vertices in G^* corresponding to the k hyperedges with minimum union in G now represent the best unbalanced k-cut in G^*. By adding sufficiently many extra vertices, we extend this argument to computing a bisection.

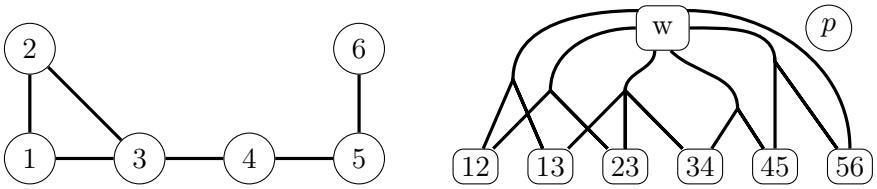

Figure 2.1.: Left: A 2-uniform instance of MkU. Right: The corresponding instance of MHB constructed through our reduction.

Given an instance of MkU $G' = (V', E')$ with $n = |V'|$ and $m = |E'|$, the reduction formally constructs the following hypergraph $G = (V, E)$. The vertex set V consists of a supervertex w and a vertex v_i for each hyperedge $h'_i \in E'$. For $v'_j \in V'$, a hyperedge h_j connects w with all vertices v_i fulfilling $v'_j \in h'_i$. Additionally, there exist p *padding vertices* u_ℓ, $1 \leq \ell \leq p$, with $p = \max\{m + 1 - 2k, 2k - m - 1\}$, where k is the number of hyperedges that must be chosen in the MkU instance. If $k > (m + 1)/2$, the padding vertices are connected with the supervertex w by infinite-cost edges. Otherwise, they are not connected to any vertex. An example of the construction is given in Figure 2.1.

As every hyperedge in the graph is incident to the supervertex w, any bisection (V_1, V_2) of G must have one side without any internal hyperedges (namely the side not containing w). Let V_1 be this side. Furthermore, let $\bar{n} = |V| = m + 1 + p$ denote the number of vertices in G. Note that $m \leq \bar{n} \leq 2m$ as $1 \leq k \leq m$.

Lemma 2.14. *An $f(|V|)$-approximation algorithm for* Minimum Hypergraph Bisection *on hypergraphs $G = (V, E)$ implies an $\mathcal{O}(f(|E'|))$-approximation algorithm for* Minimum k-Union *on hypergraphs $G' = (V', E')$, for some polynomial function f.*

Proof. We use the construction described above to obtain an instance of MHB from an instance of MkU. Let $S^* \subset E'$ be an optimal solution to the MkU instance covering $\mathrm{OPT}_{\mathrm{MkU}}$ vertices. We first claim that a bisection of G cutting $\mathrm{OPT}_{\mathrm{MkU}}$ hyperedges exists. Observe that the hyperedges of S^* correspond to k vertices of hypergraph G. These vertices are only connected to $\mathrm{OPT}_{\mathrm{MkU}}$ hyperedges (those corresponding to vertices covered by S^*), so disconnecting them from the rest of G costs at most $\mathrm{OPT}_{\mathrm{MkU}}$. If $k > (m + 1)/2$, the number of vertices in V is $\bar{n} = 2k$, so we have a bisection. If $k \leq (m + 1)/2$, the number of vertices $\bar{n} = 2(m + 1 - k)$. Adding the $p = m + 1 - 2k$ padding vertices that are not covered by any hyperedge to our k vertices from S^* gives a bisection.

Let (V_1, V_2) be the bisection of G obtained by an $f(|V|)$-approximation algorithm and let b be the number of hyperedges it cuts. Clearly $b \leq f(|V|)\,\mathrm{OPT}_{\mathrm{MkU}}$ as there exists a bisection of cost $\mathrm{OPT}_{\mathrm{MkU}}$. We show how to construct a set $S' \subset E'$ of k hyperedges

that cover at most b vertices of the MkU instance. This gives an $f(|V|) = \mathcal{O}(f(|E'|))$-approximate solution for the MkU instance, because $|V| \leq 2m = 2|E'|$ and f is at most a polynomial.

First, suppose $k > (m+1)/2$. Then both V_1 and V_2 consist of $\bar{n}/2 = k$ vertices. As the p padding vertices are connected to the supervertex with edges of infinite cost in this case, all of them must be in V_2, the part containing the supervertex. The k vertices of V_1 are therefore connected to b hyperedges in total. To see this, observe that each of these hyperedges contains the supervertex and is therefore cut by the bisection. Hence the k vertices of V_1 correspond to k hyperedges of E' covering b vertices.

Now, suppose that $k \leq (m+1)/2$. Then both V_1 and V_2 consist of $\bar{n}/2 = (m+1-k)$ vertices. Again, assume that V_1 does not contain the supervertex and observe that V_1 has no internal hyperedges. The number of padding vertices is $p = m+1-2k$, so V_1 contains at least k non-padding vertices. Choose an arbitrary set S of k non-padding vertices from V_1. The vertices of S are connected to at most b hyperedges as they, too, have no internal hyperedges. The set S therefore corresponds to a set of k hyperedges from E' that cover at most b vertices of V'. □

Theorem 2.15. *Assuming Hypothesis 2.11 (HDRH), there is no $\mathcal{O}\big(n^{1/4-\varepsilon}\big)$ approximation algorithm for* Minimum Hypergraph Bisection*, where n denotes the number of vertices of the hypergraph, for any $\varepsilon > 0$.*

Furthermore, there is no $\mathcal{O}\big(n^{\min\{\alpha/2,\alpha(1-\alpha)\}-\varepsilon}\big)$ approximation algorithm for Minimum Hypergraph Bisection *on $\Theta\big(n^\alpha\big)$-uniform hypergraphs, for any $\varepsilon > 0$.*

Proof. The lower bound of $\mathcal{O}\big(n^{1/4-\varepsilon}\big)$ follows immediately from Lemma 2.14 and the work by Chlamtáč et al. [CDM17].

For the second statement, observe that our reduction from MkU to MHB transforms a quasi α-uniform instance (H', n, m, α, r) of MkU into an instance $G = (V, E)$ of MHB. The hypergraph G has $\bar{n} = \Theta(m)$ vertices and $\bar{m} = n$ hyperedges. As the instance of MkU is quasi α-uniform, every vertex of G' has degree $\Theta(n^\alpha)$, and, given that the size r of the hyperedges is constant, $m = \Theta(n^{1+\alpha}/r) = \Theta(n^{1+\alpha})$. The hyperedges of G therefore have size $\Theta(n^\alpha) = \Theta(m^{\alpha/(1+\alpha)}) = \Theta(\bar{n}^{\alpha/(1+\alpha)})$. With $\gamma = \alpha/(1+\alpha)$, the hyperedges of G thus all have size $\Theta(\bar{n}^\gamma)$.

Lemma 2.12 implies that quasi α-uniform instances of MkU cannot be approximated up to factor $m^{\min\{\alpha/(2+2\alpha),\alpha/(1+\alpha)^2\}-\varepsilon}$. As

$$\frac{\alpha}{(1+\alpha)^2} = \frac{\alpha}{1+\alpha} \cdot \frac{1}{1+\alpha} = \frac{\alpha}{1+\alpha} \cdot \left(1 - \frac{\alpha}{1+\alpha}\right) \ ,$$

it follows that MHB with hyperedge sizes $\Theta(n^\gamma)$ cannot be approximated with factor $\bar{n}^{\min\{\gamma/2,\gamma(1-\gamma)\}}$. □

Via the reduction of Proposition 2.3, any hypergraph can be transformed into a simple graph with two weight functions. Cuts in this graph are equivalent to the hypergraph, if the balancing weight function is observed. Together with the previous theorem, this gives a lower bound on the approximation ratio of an algorithm for Minimum Vertex Bisection with two weight functions.

Corollary 2.16. *Assuming Hypothesis 2.11 (HDRH), there is no $\mathcal{O}\big(n^{1/4-\varepsilon}\big)$ approximation algorithm for doubly weighted* Minimum Vertex Bisection*, where n denotes the number of vertices of the graph, for any $\varepsilon > 0$.*

Hardness based on exponential time hypothesis

The inapproximability of MkU and, hence, MHB not only follows from HDRH. We show next that an f-approximation algorithm for these problems implies an $\mathcal{O}(f^2)$-approximation algorithm for Densest k-Subgraph (DkS). Recall that DkS asks to find a k-vertex subgraph of a given graph with maximum number of edges. The hardness of this problem has been widely studied, see, e.g., Manurangsi [Man18], Bhaskara et al. [BCV+12] and references therein. Recently, Manurangsi showed that the problem does not admit an $n^{1/(\log\log n)^c}$-approximation algorithm under the Exponential Time Hypothesis for some universal constant $c > 0$. Under the stronger GAP-ETH[1], no subpolynomial approximation ratio is possible [Man17].

Lemma 2.17. *An $f(|V|)$-approximation algorithm for* Minimum Hypergraph Bisection *implies an $\mathcal{O}\big(f^2(|V|)\big)$-approximation algorithm for* Densest k-Subgraph *for any polynomial function f.*

Proof. Given an instance $G = (V, E)$ of DkS, we first guess the number of hyperedges in the densest k-vertex subgraph of G. Call this number ℓ^*. The goal of an α-approximation algorithm to DkS is to find some k-vertex subgraph of G with at least ℓ^*/α edges.

We define the following problem $\mathcal{P}$: Find those ℓ^* *edges*, whose induced subgraph contains the minimum number of *vertices*. Let k^* be the number of vertices in an optimal solution to this new problem and observe that $k^* \leq k$. This is because the optimal subgraph of our DkS-instance is a feasible solution to problem $\mathcal{P}$ as well. A

[1]The GAP-ETH is an open hypothesis given by Dinur as follows: "For some constant $c > 0$, any algorithm that is given a 3SAT formula Φ on n variables and $\mathcal{O}(n)$ clauses, and decides between $\mathrm{sat}(\Phi) = 1$ and $\mathrm{sat}(\Phi) < 0.9$ must run in time at least 2^{cn}. (Here $\mathrm{sat}(\Phi)$ denotes the maximal fraction of satisfied clauses)." [Din16].

β-approximation algorithm to problem $\mathcal{P}$ returns a subgraph G' of G with ℓ^* edges and at most $k^*\beta \leq k\beta$ vertices.

Suppose we are given a β-approximate solution G' to $\mathcal{P}$. Then a random k-vertex subgraph of G' contains any edge of G' with probability at least $\left(\frac{k}{k\beta}\right)^2 = \frac{1}{\beta^2}$. It follows that a random k-vertex subgraph of G' contains ℓ^*/β^2 edges. This means that the random subgraph is a β^2-approximation to the densest k-subgraph. Furthermore, the random choice can be derandomized via the method of conditional expectations.

It remains to show what factor β we can achieve for problem $\mathcal{P}$. Suppose that we are given an $f(|V|)$-approximation algorithm for MHB. We claim that this implies $\beta = f(|V|)$.

Observe that $\mathcal{P}$ is in fact an instance of MkU on the dual graph of G, i.e., the instance of MkU has $|V|$ hyperedges and $|E|$ vertices. Following Lemma 2.14, an $f(|V|)$-approximation algorithm for MHB is an $\mathcal{O}(f(m))$-approximation to the MkU, where m is the number of hyperedges in the instance. Here, $m = |V|$. This proves $\beta = f(|V|)$. □

Corollary 2.18. *Assuming the ETH, there is no efficient approximation algorithm for* Minimum Hypergraph Bisection *with approximation ratio* $n^{1/(\log\log n)^c}$ *for some universal constant* $c > 0$.

Proof. Follows from Lemma 2.17 and [Man17]. □

2.3. Trees for Vertex Cuts and Hypergraph Cuts

This section considers the more general challenge of representing the complete cut structure of a hypergraph by a tree. We further extend this to vertex cuts in graphs and give a construction of a vertex cut tree with quality $\tilde{\mathcal{O}}(\sqrt{W})$, where W is the total vertex weight. We then explore the limits in constructing such a tree by showing lower bounds on the quality of cut trees. Observe that the lower bounds shown in this section do not rely on any assumptions from computational complexity. This is partly due to the fact that we only exclude the existence of algorithms via a specific *technique*, instead of showing a lower bound over all algorithms.

2.3.1. Constructing vertex cut trees

This section demonstrates the construction of a dominating vertex cut tree of quality $\tilde{\mathcal{O}}(\sqrt{W})$ for node-weighted graphs of total weight W. We assume the minimum weight of a vertex to be 1 as can be achieved in many scenarios through rescaling.

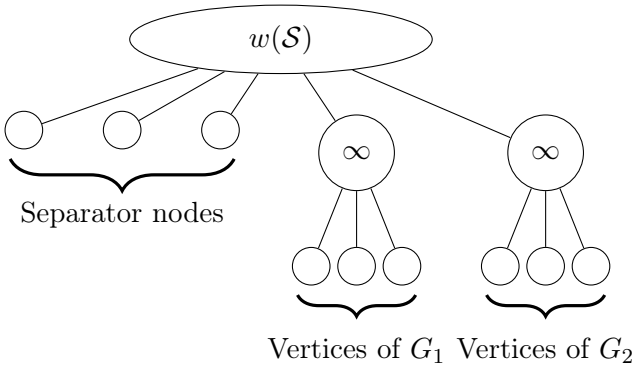

Figure 2.2.: Construction of the vertex cut tree.

Construction

The construction of our vertex cut tree employs a recursive partitioning of the graph using min-ratio vertex cuts. We define the weighted version of a min-ratio vertex cut of $G = (V, E, w)$ as a vertex separator (A, B, S) of minimum sparsity

$$\frac{w(S)}{\min\{w(A), w(B)\} + w(S)} \ .$$

Let $\mathcal{A}$ be an α-approximation algorithm for finding min-ratio vertex cuts. There exist polynomial-time algorithms with $\alpha = \mathcal{O}(\sqrt{\log n})$ [FHL08, ACMM05].

Use Algorithm $\mathcal{A}$ to find the min-ratio vertex cut (A, B, X) in the graph. Let $G_1^X, G_2^X, \ldots$ be the connected components after removing the separator X. Repeat the process on each subgraph until no cut of sparsity less than $\alpha f(W)$ can be found for some function f that we chose later. This implies that no cut of sparsity less than $f(W)$ exists.

Let $G_1, G_2, \ldots$ denote the subgraphs remaining at the end of this process with $G_i = (V_i, E_i)$ and let $\mathcal{S}$ denote the union of all separators found. The cut tree is constructed by first adding a root vertex r of weight $w(\mathcal{S})$. For any vertex $s \in \mathcal{S}$ the root obtains a child of weight $w(s)$. Further, the root obtains a child t_{G_i} of infinite weight for any subgraph G_i. Any vertex $v \in V_i$ is added as a leaf of weight $w(v)$ to t_{G_i}. A sketch of the construction can be seen in Figure 2.2.

Analysis

Our analysis proceeds in two parts. We first show that the tree constructed according to the above algorithm is dominating (independent of the choice of f). We then show that

for an appropriate choice of f, the tree has quality $\mathcal{O}(\sqrt{\alpha \log nW})$.

Lemma 2.19. *For any two disjoint sets $A, B \subset V$, we have $\gamma_G(A,B) \leq \gamma_T(A,B)$.*

Proof. Let X be some minimum vertex separator of A and B in the tree of non-infinite weight. We construct a set $X' \subset V$ of equal weight that separates A and B in the graph.

First suppose that X does not contain the root r. Then X only contains vertices of V and we set $X' = X$. Additionally, any pair of vertices $a, b \in V$ is only separated by X in T if either a or b belongs to X. It follows that either $A \subset X$ or $B \subset X$ and therefore $X' = X$ separates A and B in G.

If X contains the root r, let $X' = (X \cap V) \cup \mathcal{S}$, i.e., the cut in the graph is the separator together with the vertices of $X \cap V$. Let $a \in A$, $b \in B$ and we show that X' separates a and b in G. If either $a \in X'$ or $b \in X'$, this is obvious, so suppose neither belong to X'; in particular, a and b are not in the separator $\mathcal{S}$. They furthermore do not belong to the same subgraph G_i, as X separates them in T. This means that a and b belong to different subgraphs G_i. As the vertices of $\mathcal{S} \subseteq X'$ separate all subgraphs G_i by construction, it follows that a and b are separated by X'. □

Lemma 2.20. *If G is connected, we can choose function f so that for any two disjoint sets $A, B \subset V$,*

$$\gamma_T(A,B) \leq \mathcal{O}(\sqrt{W\alpha \log n})\gamma_G(A,B) \ .$$

Proof. In the following we construct a vertex cut X separating A and B in T.

First, we include the root r of T into this cut. We derive a bound on the weight $w(r)$ as follows. The weight of the root is the total weight $w(\mathcal{S})$ of the vertices in the separator $\mathcal{S}$. For any separator (A_i, B_i, S_i) found during the construction let A_i be the side of the cut that has the smaller number of vertices (not necessarily the smaller weight). We have

$$\frac{w(S_i)}{w(A_i) + w(S_i)} \leq \frac{w(S_i)}{\min\{w(A_i), w(B_i)\} + w(S_i)} \leq \alpha \cdot f(W) \ .$$

The second inequality follows from the construction of the algorithm. This gives

$$\begin{aligned} w(S_i) &\leq \frac{1}{1 - \alpha f(W)} \alpha f(W) \cdot w(A_i) \\ &\leq 2\alpha f(W) \cdot w(A_i) \ , \end{aligned}$$

for sufficiently large n if we choose f such that $\alpha f(W) = o(1)$. We amortize the weight $w(S_i)$ of the separator against the weight $w(A_i)$ of the vertices in A_i. Since a vertex can

be on the smaller side of the cut at most $\log n$ times we get that

$$w(\mathcal{S}) = \sum_i w(S_i) \leq 2\alpha f(W) W \log n \ .$$

Since the cost of the optimum cut is at least 1 this gives

$$w(r) = w(\mathcal{S}) \leq 2\alpha f(W) W \log n \cdot \gamma_G(A, B) \ . \tag{2.1}$$

In addition to the root r we add some leaf vertices for every subgraph $G_i = (V_i, E_i)$ to the cut X. Let $A_i = A \cap V_i$ and $B_i = B \cap V_i$ (note that these sets might be empty). If $w(A_i) \leq w(B_i)$ we add the leaf vertices corresponding to A_i, otherwise we add those corresponding to B_i. This forms an (A, B)-cut in T: a vertex $a \in A_i$ is separated from vertices in B of other subgraphs $G_{i'}$ because of the root r, and it is separated from the vertices in B of its own subgraph G_i because either A_i or B_i is in the cut. We use X_i to denote the set of vertices that were added in this step for subgraph G_i.

Let X_i^* denote the intersection of the optimal (A, B)-separator with the subgraph G_i. Since the sparsity of any vertex separator in G_i is at least $f(W)$ it follows that

$$\frac{w(X_i^*)}{w(X_i \cup X_i^*)} \geq f(W) \ .$$

To see this, recall that X_i is either A_i or B_i, whichever has the smaller weight, hence X_i^* separates X_i from the rest of the subgraph. By rearranging this inequality we obtain $w(X_i) \leq w(X_i \cup X_i^*) \leq f(W)^{-1} w(X_i^*)$. As the subgraphs G_i are disjoint, taking the sum over all i gives

$$\sum_i w(X_i) \leq f(W)^{-1} \sum_i w(X_i^*) \leq f(W)^{-1} \gamma_G(A, B) \ . \tag{2.2}$$

Combining equations (2.1) and (2.2) gives that the cut X we constructed fulfills

$$w(X) \leq (f(W)^{-1} + 2\alpha f(W) W \log n) \cdot \gamma_G(A, B) \ .$$

We now choose $f(W) = 1/\sqrt{2\alpha W \log n}$ to balance the two terms of the above sum. Observe that, as $\alpha \in \mathcal{O}(\sqrt{\log n})$, we fulfill $\alpha f(W) = o(1)$. With our choice of f, we obtain $w(X) \leq \mathcal{O}(\sqrt{W \alpha \log n})$. □

Combining Lemma 2.19 and Lemma 2.20 with $\alpha \in \mathcal{O}(\sqrt{\log n})$ gives the following theorem.

Theorem 2.21. *For a graph G with total vertex weight W, there is a polynomial-time construction of a vertex cut tree for approximating vertex cuts with quality $\mathcal{O}(\log^{3/4} n)\sqrt{W}$.*

Application to vertex bisection

We demonstrate how to apply Theorem 2.21 in order to find a minimum vertex bisection.

A vertex bisection of a graph $G = (V, E)$ is a coloring of the graph vertices using *three* colors black, white and gray. The number of black vertices is at most $|V|/2$, an so is the number of white vertices. The cost of the bisection is the number of gray vertices. No edge of the graph connects a black vertex and a white vertex.

Given graph G, construct a vertex cut tree $T = (V_T, E_T)$ as described in the previous section. Let $\mathrm{bw} : V_T \to \{0, 1\}$ be a function assigning *balance weight* to the vertices of the tree. A vertex $v \in V$ of the original graph has balance weight 1. A vertex $v \in V_T \setminus V$ has balance weight 0. Observe that any vertex bisection in G induces a coloring of the tree vertices, so that any path between a black vertex and a white vertex must contain a gray vertex. In this coloring, the total balance weight of black vertices is at most $n/2$; similarly, the total balance weight of white vertices is at most $n/2$. Again, the cost of the partition is the total cost of the gray vertices.

Tthe minimum-cost coloring of the vertices of T respecting these two conditions can be found through a dynamic program. We say a coloring of a subtree is *valid* if any path between a black and a white vertex contains a gray vertex.

For any tree vertex v, $C_v^k(p, q, x)$ is the cost of the minimum-cost valid coloring of v and the subtrees of its k first children, where the color of v is $x \in \{b, g, w\}$, p is the total balance weight of the white vertices, and q is the total balance weight of the black vertices. If v is a leaf, then $C_v^0(0, \mathrm{bw}(v), b) = 0$, $C_v^0(\mathrm{bw}(v), 0, w) = 0$ and $C_v^0(0, 0, g) = c(v)$, all other values are infinite. For any internal vertex, $C_v^1(p, , qg) = \min_{x\in\{g,w,b\}}\{C_u^{k_u}(p, q, x)\} + c(v)$, where u is the first child of v and k_u its number of children. Similarly, we set $C_v^1(p, q, w) = \min_{x\in\{g,w\}}\{C_u^{k_u}(p-\mathrm{bw}(v), q, x)\}$ and we set $C_v^1(p, q, b) = \min_{x\in\{g,b\}}\{C_u^{k_u}(p, q-\mathrm{bw}(v), x)\}$. For $k > 1$, let u denote the k-th child of v and k_u the number of children of u, then

$$C_v^k(p, q, g) = \min_{\substack{p=p_u+p' \\ q=q_u+q' \\ x_u\in\{b,w,g\}}} \{C_v^{k-1}(p', q', g) + C_u^{k_u}(p_u, q_u, x_u)\} \ ,$$

$$C_v^k(p, q, w) = \min_{\substack{p=p_u+p'+\mathrm{bw}(v) \\ q=q_u+q' \\ x_u\in\{g,w\}}} \{C_v^{k-1}(p', q', w) + C_u^{k_u}(p_u, q_u, x_u)\} \ ,$$

$$C_v^k(p,q,b) = \min_{\substack{p=p_u+p' \\ q=q_u+q'+\mathrm{bw}(v) \\ x_u\in\{g,b\}}} \{C_v^{k-1}(p',q',b) + C_u^{k_u}(p_u,q_u,x_u)\} \ .$$

The algorithm outputs the partition associated with the smallest entry $C_r^{k_r}(p,q,x)$, where $p,q < |V|/2$, r is the root of the tree, and $x \in \{b,g,w\}$ is arbitrary.

Corollary 2.22. *There is an $\mathcal{O}\big(\sqrt{n}\log^{3/4} n\big)$ approximation algorithm for* Minimum Vertex Bisection *in an n-vertex graph $G=(V,E)$.*

Proof. The correctness of the dynamic programming scheme follows by induction over the depth of a vertex. The quality of the cut tree is $\mathcal{O}(\log^{3/4} n)\sqrt{W}$, where W is the total vertex weight of graph G by Theorem 2.21. □

Corollary 2.23. *There is an $\mathcal{O}\big(\sqrt{nd_{\mathrm{avg}}}\log^{3/4} n\big)$ approximation algorithm for* Minimum Hypergraph Bisection *in an n-vertex hypergraph $G=(V,E)$, where d_{avg} is the average vertex degree in G.*

Proof. This follows by applying the reduction from Proposition 2.2. In the tree, we must slightly change the definition of balance weight, as only vertices that correspond to vertices in the original hypergraph instance obtain balance weight 1.

If the solution computed on the tree places a vertex $v \in V$ of the original hypergraph in the separator, we modify the solution by adding all hyperedges incident to v into the cut. This only decreases the cut size and we may place v in an arbitrary part of the bisection. □

Remark. It seems tempting to use the reduction from Proposition 2.3 in the above corollary and obtain an $\mathcal{O}\big(\sqrt{n}\log^{3/4} n\big)$ approximation algorithm for Minimum Hypergraph Bisection. Unfortunately, this is not possible. Recall that Proposition 2.3 constructs a graph with *two* independent vertex weight functions. One of the weight functions gives the balancing weight of the vertex, the other one gives its cut weight. The construction of our cut tree T, however, cannot properly distinguish two weight functions and the proof of its quality fails. More specifically, the proof of Lemma 2.20 fails, as the optimal cut in a subtree cannot simply take all vertices from the smaller side, as both sides possibly have infinite cut weight.

2.3.2. Lower bounds

We complement the positive result from the previous section by various lower bounds for approximating vertex cuts or hypergraph cuts by cut trees. We first show that in

order to get any reasonable approximation guarantee one needs to consider vertex cut trees instead of edge cut trees. This is in strong contrast to ordinary graphs where edge cut trees already give a polylogarithmic guarantee, see, e.g., Räcke and Shah [RS14]. We then show that even vertex cut trees will not yield such guarantees by giving an $\mathcal{O}(\sqrt{n})$ lower bound for hypergraph cuts and an $\mathcal{O}(n^{1/3})$ lower bound for vertex cuts.

Edge cut trees for hyperedge cuts

Theorem 2.24. *For any $n \geq 3$ there exists a hypergraph on n vertices so that the quality of any edge cut tree is at least $n/8$.*

Proof. Consider a hypergraph $H = (V, E_H)$ over n vertices whose only hyperedge spans all vertices. Let $T = (V_T, E_T)$ be a rooted edge cut tree for hypergraph H. Without loss of generality, every edge of T defines a cut of weight 1 in the hypergraph, thus we may assume that all edges of the tree have weight 1. Furthermore, we may assume that every internal vertex of the tree except the root has at least degree 3, otherwise we can remove that vertex without changing the weight of any cut. It follows that T has less than $2n$ edges.

We now derive our contradiction. Let A, B be two different subsets of V with $A \neq V \setminus B$. Let $F_A, F_B \subset E_T$ be the minimum-size sets of edges separating A from $V \setminus A$ in T and B from $V \setminus B$ in T, respectively. Clearly F_A and F_B must be different, as a smaller cut for either A or B could otherwise be found. This means that, for every nontrivial subset of hypergraph vertices $\emptyset \neq A \subsetneq V$, there must be a unique set of edges that is the minimum cut of A and $V \setminus A$ in the tree. We know that there are $2^n - 2 > 2^{n-1}$ proper, nonempty subsets of V . After excluding half of them for being a complement, 2^{n-2} remain, all of which must be well-represented by T.

As $|E_T| < 2n$, the number of subsets of E_T of size at most $n/8$ is less than

$$\sum_{i=1}^{n/8} \binom{2n}{i} \leq \frac{n}{8}\binom{2n}{n/8} \leq \frac{n}{8}\left(e\frac{2n}{n/8}\right)^{n/8} < \frac{1}{4}\left(32e\right)^{n/8} < \frac{1}{4}\left(2^{7/8}\right)^{n} < 2^{n-2} \ .$$

The third inequality holds as $(n/2) < 2^{n/8}$ for $n \geq 3$. This means that the number of subsets with cost at most $n/8$ is less than 2^{n-2}. Hence, there must be at least one nontrivial vertex set A whose cut cost in the tree is at least $n/8$, which proves the lemma. □

Observe that the above proof can also be generalized to show that for any polynomial-sized family of edge cut trees, there must be at least one cut in the hypergraph that each

one of the trees approximates with factor $\Omega(n)$. Furthermore the theorem implies that a simple, asymptotically optimal construction of an edge cut tree is the following. Suppose that the hypergraph is connected, then connect all vertices with a simple path. Every edge of the path is weighted with the number of hyperedges containing it. The cut cost of the path now clearly dominates the hypergraph and each hyperedge contributes at most its size to the cut cost of the path. This gives quality $h_{\max} \leq n$.

Vertex cut trees for hyperedge cuts

The bad example from the previous section can be perfectly represented by a *vertex cut tree*, namely a star on all vertices. The following theorem gives a lower bound on the quality of vertex cut trees for approximating hyperedge cuts in a hypergraph H.

Theorem 2.25. *There exists a hypergraph on n vertices such that the quality of any vertex cut tree is $\Omega(\sqrt{n})$.*

Proof. Consider a hypergraph $H = (V, E_H)$ with a top vertex v connected to n vertices $U = \{u_1, \ldots, u_n\}$ with simple edges of weight 1, as well as a hyperedge of weight $\sqrt{n}$ spanning all vertices u_i. Figure 2.3 shows a sketch of graph H.

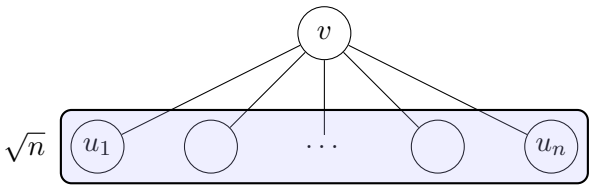

Figure 2.3.: Bad example for approximating hyperedge cuts via vertex cut trees.

The cut size of any non-empty set $S \subsetneq U$ is $\delta_H(S) = \sqrt{n} + |S|$, as both the hyperedge and all edges connecting S to the top vertex v must bet removed. The cut size of any set $S' \cup \{v\}$, $\emptyset \neq S' \subsetneq U$ is $\delta_H(S' \cup \{v\}) = \sqrt{n} + n - |S'|$. It follows that any set S with $1 \leq \ell < n/2$ vertices from U has $\delta_H(S) \geq \sqrt{n} + \ell$.

Let T be a dominating vertex cut tree of H. Fix a root of T so that none of its children have strictly more than $n/2$ vertices from U in its subtree. Also fix an arbitrary order among the children of each node.

Suppose that some node x of this tree has ℓ vertices of U in its subtree. We claim that x corresponds to a cut in the graph with at least $\ell/3$ vertices of U on both sides of the cut. This is evident if $\ell \leq n/2$: node x separates the ℓ vertices in its subtree from the $n - \ell > \ell/3$ vertices outside its subtree. If $\ell > n/2$, recall that none of the children of x have more than $n/2$ vertices of U in its subtree. This means that x corresponds to a cut

that partitions the graph into subgraphs, so that no subgraph contains more than $n/2$ vertices of U. We may therefore group the subgraphs in such a way that both groups contain at least $n/3 \geq \ell/3$ vertices of U. As the tree is dominating, node x must therefore have weight at least $w(x) \geq \max\{\sqrt{n}, \ell/3\}$.

We now explicitly construct a set S whose cut size is $\Omega(\sqrt{n})$. Suppose without loss of generality that an in-order traversal of the tree T visits the vertices of U in the order $u_1, u_2 \dots u_n$. Let $k = \lfloor\sqrt{n}\rfloor = \Theta(\sqrt{n})$ and $S = \{s_1, \dots, s_k\}$ be the set $\{u_k, u_{2k}, \dots, u_{k^2}\}$ with $s_i = u_{ik}$ for $i \in [k]$. As $|S| = k < n/2$, the cut size of S in H is $\delta_H(S) = k+\sqrt{n} \leq 2\sqrt{n}$. We show that the vertex cut cost in the tree $\gamma_T(S, V \setminus S)$ is $\Omega(n)$.

Fix a cut $X = \{x_1, \dots, x_{|X|}\}$ of S and $V \setminus S$ in T. As any node of T has weight at least $\sqrt{n}$, we assume that $|X| \leq k/2$ as otherwise the total weight of X is $\Omega(n)$ immediately. Define another subset of $R \subset U$ disjoint from S to be $R = \{r_1, \dots, r_k\} = \{u_{k/2}, u_{k/2+k}, \dots, u_{k^2-k/2}\}$ with the correspondence $r_i = u_{ik-k/2}$. We assume here that k is divisible by 2, otherwise let $r_i = u_{ik-(k-1)/2}$, which does not change our results.

We count for every node x_j the number p_j of (s_i, r_i) pairs it separates. More precisely, let p_j be the number of pairs (s_i, r_i) for which x_j lies on the unique path connecting them in T. As X separates the sets S and R, we know

$$\sum_{j=1}^{|X|} p_i \geq k \ .$$

Moreover, the weight of a node x_j is roughly k times p_j:

Claim 2.26. $w(x_j) \geq \frac{k}{6}(p_j - 1)$.

Proof. If $p_j = 1$, the right-hand side is 0, so the claim holds. Assume that $p_j > 1$. Recall that an in-order traversal of the tree visits the vertices of U in the order $u_1, \dots, u_n$. This means that if the subtree of x_j contains vertices u_i and $u_{i'}$ with $i < i'$, it also contains all vertices $u_{i''}$ for $i \leq i'' \leq i'$. For any pair (r_i, s_i) that x_j separates, at least one of r_i and s_i must lie in the subtree of x_j. By definition of r_i and s_i, this means that the subtree of x_j contains at least one of $u_{ik-k/2}$ and u_{ik}. Let i, i' be the smallest and largest integers, respectively, so that x_j separates (r_i, s_i) and $(r_{i'}, s_{i'})$. With our above observation, this means that the subtree of x_j contains at least all nodes u_t with $ik \leq t \leq i'k - k/2$ i.e., it contains at least $k(i' - i) - k/2$ nodes.

By definition of p_j it is at most $i' - i + 1$, so the number of nodes in the subtree of x_j is at least

$$k(i' - i) - \frac{k}{2} \geq k(p_j - 1) - \frac{k}{2} \geq \frac{k}{2}(p_j - 1) \ .$$

As any node with ℓ nodes of U in its subtree has at least weight $\ell/3$, the weight of x_j must be at least $\frac{k}{6}(p_j - 1)$. □

The total weight of X is therefore at least

$$\sum_{j=1}^{|X|} w(x_j) \geq \sum_{j=1}^{|X|} \frac{k}{6}(p_j - 1) = \frac{k}{6}\sum_{j=1}^{|X|} p_j - \frac{k}{6}|X| \geq \frac{k^2}{6} - \frac{k^2}{12} = \frac{k^2}{12} \ .$$

The first step uses Claim 2.26 and the second inequality uses the assumption that $|X|$ contains at most $k/2$ vertices. As $k = \Theta(\sqrt{n})$, the total weight of X is $\Omega(n)$. □

Note that the same result holds for unweighted hypergraphs. To see this, replace the weighted hyperedge with $\lfloor\sqrt{n}\rfloor$ different hyperedges, each spanning $n - 1$ vertices of the set U.

Vertex cut trees for weighted vertex cuts

The result from the previous section also yields a lower bound for vertex cut trees approximating weighted vertex cuts. The proof is based on the graph $G_H = (V_H, E_H)$ shown in Figure 2.4. Graph G_H is precisely the graph constructed by applying the reduction from Proposition 2.2 to the hypergraph H from the previous section. It consists of a vertex t of weight $\sqrt{n}$ (representing the hyperedge from H), connected to vertices $u_1, \ldots, u_n$ of weight $\sqrt{n} + 1$. Each of the u_i is connected to a vertex w_i of weight 1 (representing the edges of H). Finally, all w_i are connected to a vertex v of weight n. Graph G_H has $N = 2n + 2$ vertices of total weight $2n + n(\sqrt{n} + 1) + \sqrt{n} = \Theta(N\sqrt{N})$.

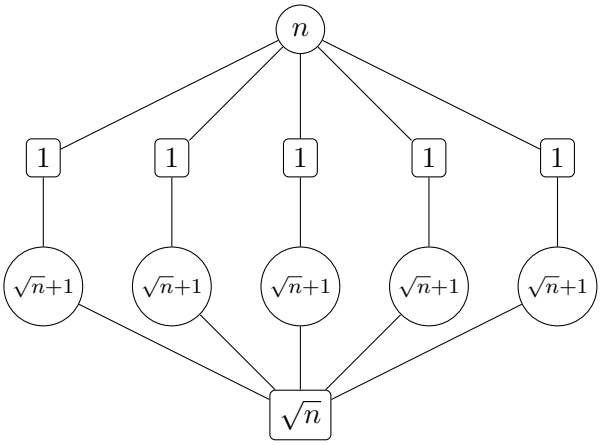

Figure 2.4.: Bad example for approximating weighted vertex cuts via vertex cut trees.

Theorem 2.27. *There exists a vertex-weighted graph on n vertices such that the quality of any vertex cut tree for approximating vertex cuts is $\Omega(\sqrt{n})$.*

Proof. Let T be a vertex cut tree for G_H. We first show that T is also a vertex cut tree for hypergraph $H = (V, E)$ considered in the previous section, shown in Figure 2.3. For any two disjoint vertex sets $A, B \subset V$, $\delta_H(A, B) = \gamma_{G_H}(A, B)$ by Proposition 2.2. As the vertex set of G_H is a superset of V, this means that T is indeed a vertex cut tree for H. By Theorem 2.25, T has quality $\Omega(\sqrt{|V|})$ for approximating hyperedge cuts in H. As the number of vertices in G_H is $N = 2|V| + 2$, it follows that T is a vertex cut tree of quality $\Omega(\sqrt{N})$ for G_H. □

Vertex cut trees for unweighted vertex cuts

We now plan to adapt the analysis from the previous section to unweighted vertex cuts. This is not straightforward as the lower bound we show relies heavily on the fact that the graph G_H is weighted.

For simplicity of exposition, let $k = \sqrt{n}$ be an integer. The graph G_H shown in Figure 2.4 is transformed as follows. Any vertex of weight z is transformed into a clique C_z on z vertices. Edges between formerly weighted vertices in G_H are transformed into complete bipartite subgraphs between these cliques. We also change the vertex weight from $\sqrt{n} + 1$ to $\sqrt{n}$ compared to the previous section, hence the number of vertices is $N = k^3 + 2k^2 + k$. Let U_i denote the clique resulting from the vertex u_i in G_H. The resulting graph G is shown in Figure 2.5.

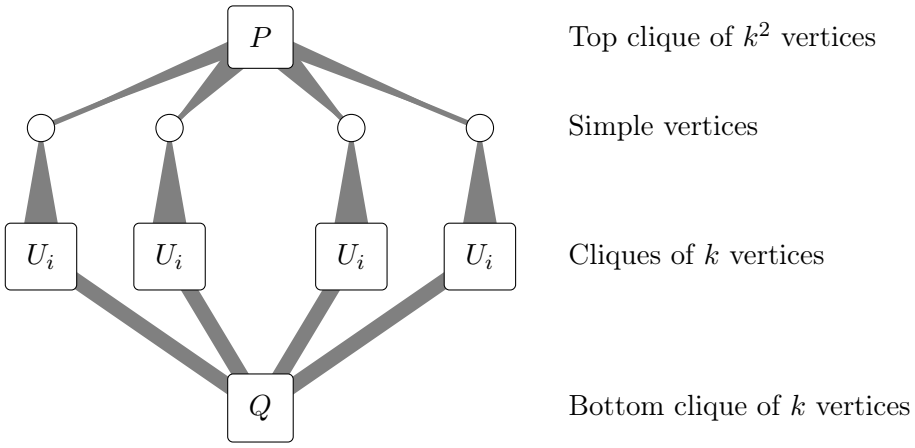

Figure 2.5.: Bad example for approximating unweighted vertex cuts via vertex cut trees.

We plan to adapt the proof of theorems 2.25 and 2.27. This proof is based on the observation that for any tree, there must exist k vertices u_i that are *spread apart* in the

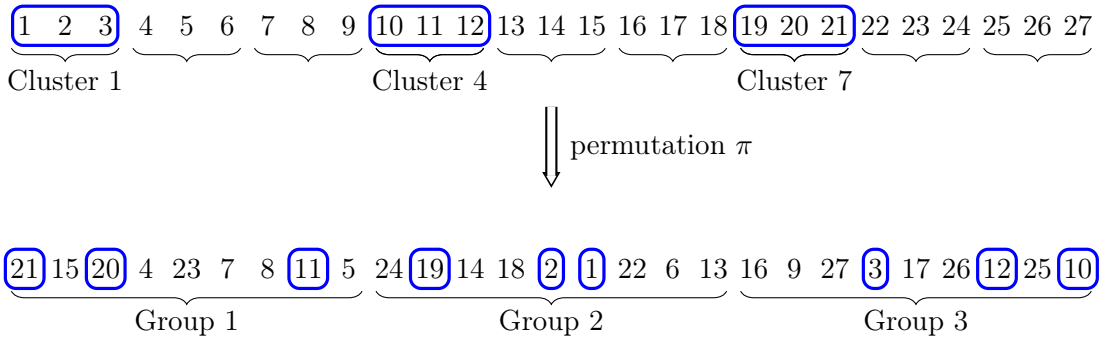

Figure 2.6.: Choose clusters 1, 4 and 7, then 3 integers from every group have been chosen.

tree and, therefore, separating these vertices in the tree is costly. When the vertices u_i are now cliques U_i, the vertices of the clique may not be concentrated in a vertex cut tree. Nevertheless, we show that there are k cliques U_i whose combined vertex set is well-spread over the tree.

In the unweighted graph, call the clique vertices U_i *core vertices*; enumerate them so that core vertices of the same clique are consecutive. Given any vertex cut tree, an in-order traversal of the tree induces a *permutation* of the core vertices. We first show that, for any permutation, there is a choice of core vertices from $\Theta(k)$ cliques, such that the following holds: After applying the permutation, there exist $\Theta(k)$ disjoint intervals of length k^2, in which $\Theta(k)$ chosen core vertices are found. This means that we can choose cliques such that the vertices from these cliques are sufficiently spread apart in the tree.

We formalize the technical observation as follows. Divide the set $L = \{1, \dots, k^3\}$ into k^2 *clusters* of k consecutive integers $\{(i-1)k+1, \dots, ik\}$ for $i = 1, \dots, k^2$. Given some permutation π of L, we define *groups*. Intuitively, a group is a set of n consecutive integers in the image of π. Formally, group j is $\{\pi^{-1}((j-1)k^2+1), \dots, \pi^{-1}(jk^2)\}$ for $j = 1, \dots, k$. We want to choose (the integers of) $\Theta(k)$ clusters, so that each group contains $\Theta(k)$ integers. An example for $k = 3$ is shown in Figure 2.6.

Lemma 2.28. *For any permutation π there exists a choice of $2k$ clusters such that there are at least $k/9$ groups from which $\Theta(k)$ integers have been chosen.*

Proof. Choose $2k$ clusters uniformly at random without replacement. We show that in expectation at least $k/9$ groups have $\Theta(k)$ integers chosen from them. This implies the lemma.

Let Y_j be the number of chosen integers from group j, and let X_j be the indicator variable of the event that $k/2 \leq Y_j \leq 7k/2$. We plan to show $\mathbb{E}[\sum_{j=1}^{k} X_j] \geq k/9$, it

therefore suffices to show for any j

$$\mathbb{E}[X_j] = \Pr[k/2 \leq Y_j \leq 7k/2] \stackrel{!}{\geq} 1/9 \ .$$

Observe that Y_j follows a weighted hypergeometric distribution: Let C_i denote the indicator variable that cluster i has been chosen and c_{ij} the number of integers from cluster j that the permutation π maps to group j. The clusters are chosen without replacement and $Y_j = \sum_{i=1}^{k^2} c_{ij} C_i$ counts the total weight (number of integers) obtained. It follows that the expectation of Y_j is $2k^2$. To see this, observe that $\mathbb{E}[C_i] = 2k/k^2 = 2/k$ and $\sum_{i=1}^{k^2} c_{ij} = k^3$.

We now argue that the variance of Y_j is at most $2k^2$. With Cov denoting the covariance,

$$\operatorname{Var}[Y_j] = \sum_{i=1}^{k^2} c_{ij}^2 \operatorname{Var}[C_i] + 2 \sum_{1 \leq i < \ell \leq k^2} c_{ij} c_{\ell j} \operatorname{Cov}[C_i, C_\ell] \ .$$

The variance of indicator variable C_i is $\frac{2}{k}(1 - \frac{2}{k}) = (2k-4)/k^2 < 2/k$. Furthermore, C_i and C_ℓ are negatively correlated for $i \neq \ell$, as sampling is without replacement: Choosing C_i reduces the probability that C_ℓ is chosen as well. It follows that the covariance is negative, so

$$\operatorname{Var}[Y_j] \leq \sum_{i=1}^{k^2} c_{ij}^2 \operatorname{Var}[C_i] < \frac{2}{k} \sum_{i=1}^{k^2} c_{ij}^2 \leq 2k^2 \ .$$

The last step follows as k^2 integers are mapped to group j in total, at most k from each cluster; so the sum $\sum_{i=1}^{k^2} c_{ij}^2$ is at most $(\max_i c_{ij}) \cdot (\sum_{i=1}^{k^2} c_{ij}) = k \cdot k^2$.

Chebychev's inequality states with $\mu = \mathbb{E}[Y_j]$ and $\sigma^2 = \operatorname{Var}[Y_j]$ for any $t > 0$

$$\Pr[|Y_j - \mu| \leq t] \geq 1 - \frac{\sigma^2}{t^2} \ .$$

We have $\sigma^2 \leq 2k^2$, $\mu = 2k$ and we choose $t = 3k/2$. The inequality then yields

$$\Pr[k/2 \leq Y_j \leq 7k/2] \geq 1 - \frac{2k^2}{(3k/2)^2} = 1 - 8/9 = 1/9 \ . \qquad \square$$

We now give a proof of the theorem.

Theorem 2.29. *There exists an unweighted graph on n vertices such that the quality of any vertex cut tree for approximating vertex cuts is $\Omega(n^{1/3})$.*

Proof. We use the graph G defined in the beginning of the section. The vertices are named as in Figure 2.5. The clique of k^2 vertices is the *top clique* P, the clique of k

vertices connected to all other cliques of k vertices the *bottom clique* Q. The single vertex connected to all vertices of clique U_i is w_i.

In order to separate any ℓ cliques U_i from the rest of G, one can remove the bottom clique Q and the vertices w_i. The cut size is therefore $\gamma_G(U') \leq k + \ell$, where U' is the union of the cliques we remove.

Let T be a dominating vertex cut tree of G and let $w(x)$ denote the weight of tree node x. Fix a root of T so that none of its children have more than $k^3/2$ core vertices in its subtree. We first show that every node x of this tree with $j \geq k$ core vertices in its subtree must have weight at least $\max\{k, j/(3k)\}$. If $j < 3k^2$, node x defines a cut in the graph with both sides having at least k core vertices. For any set of k disjoint pairs of core vertices, there exist k vertex-disjoint paths connecting them via the bottom clique. All of these paths must be cut by node x so its weight must be at least k. If $j \geq 3k^2$, observe that x defines a cut in the graph with both sides having at least $j/3$ core vertices. Pick $j/(3k)$ cliques U_i from each side with no clique being chosen twice, and then one vertex per clique. This gives $j/(3k)$ pairs of vertices separated by x. For each pair, there is a path connecting the two vertices via the top clique and we can choose these paths so that they are all vertex disjoint. As node x interrupts all of the paths, its weight must be at least $j/(3k)$.

Enumerate the core vertices of the tree via an in-order traversal. The vertices $jk^2 + 1, \ldots, (j+1)k^2$ form a *group* in the tree, for $j = 1, \ldots, k$. Choose $2k$ cliques U_i so that at least $k/9$ groups have $\Theta(k)$ of their vertices chosen. These vertices form the set S. An appropriate choice is possible, as shown by Lemma 2.28. The cut cost of S in the graph is $\mathcal{O}(k)$, as argued above. We prove that the cut cost of S in the tree is $\Omega(k^2)$. In the following we refer to the vertices in S as S–vertices and to those in $U \setminus S$ as U–vertices. Fix a separator X of S in T. We first show that the cut cost of X is large if only few U–vertices can reach the root.

Claim 2.30. *If in a subtree T' at least j U–vertices are separated from the root, the cut cost inside T' is at least $j/(3k)$.*

Proof. The proof is by induction over the height of T'. The claim holds if T' is just a leaf vertex. If a U–vertex is cut away from the root it means that the leaf vertex is in the cut and the cost is at least $1 \geq 1/(3k)$.

For the induction step consider a root vertex $r_{T'}$ with children $c_1, \ldots, c_s$. Assume that we aim to separate ℓ_i U–vertices of the subtree of c_i from the root. If vertex $r_{T'}$ is part of the separator, the claim holds as $w(r_{T'}) \geq j/(3k)$. Otherwise, we have to pay at least $\ell_i/(3k)$ in each subtree by induction hypothesis which sums up to the same cost. □

Removing node set X from the tree partitions T into connected components. If the subtree of the (former) root contains at least one S–vertex, the cost of X is $\Omega(k^2)$ by Claim 2.30. This follows, as all $k^3 - 2k^2$ U–vertices are then disconnected from the root. We therefore assume that no S–vertex is in the connected component of the root. For tree node x, let S_x denote the set of S–vertices in the subtree of x. Observe that with our assumption $\bigcup_{x \in X} S_x = S$.

We show below that $w(x)$ is at least $|S_x|/6$ for any $x \in X$. This gives that the total weight of set X is

$$\sum_{x \in X} w(x) \geq \frac{1}{6} \sum_{x \in X} |S_x| \geq \frac{1}{6}|S| = k^2/2 \ .$$

The second step follows from the fact that $\bigcup_{x \in X} S_x = S$. The cut cost of S in T is therefore $\Omega(k^2)$ while its cut cost in G is only $\mathcal{O}(k)$. As the number of nodes in G is $n = \Theta(k^3)$ this gives a lower bound of $\Omega\big(n^{1/3}\big)$ in the quality of the tree.

Claim 2.31. *$w(x) \geq |S_x|/6$ for any $x \in X$.*

Proof. Recall that $|S_x| \leq |S| = 2k$ and $w(x) \geq \max\{j/(3k), k\}$, if $j \geq k$, where j denotes the number of vertices in the subtree of x. The claim therefore holds if $j \geq k$. Now suppose that the subtree of x contains $j < k$ core vertices. We show that $w(x) \geq j$. To see this, observe that x must separates a clique U_i. Pair the j vertices in U_i with j vertices in some other clique and connect the pairs by mutually vertex-disjoint paths. As node x interrupts all of these paths in the tree, its weight must be at least j. It follows that $w(x) \geq j \geq |S_x|$. □

□

3. Generalized Reordering Buffer Management

This chapter presents our results for Generalized Reordering Buffer Management. Our main result is an online algorithm for the problem that achieves competitive ratio $\mathcal{O}(\log k(\log k+\log\log b))$. The algorithm is based on a new linear programming formulation of the problem which is shown in the first section of the chapter. The next section shows how this LP can be used for designing an online algorithm. The analysis of that algorithm is split into sections 3.3 and 3.4. The first part of the analysis focuses on explaining how the different elements of the algorithm harmonize. The second part demonstrates in detail how the algorithm handles $k > 1$ servers, as opposed to the (already nontrivial) classical scenario of a single server that is RBM.

3.1. The Linear Programming Relaxation

Recall that we define GRBM as the problem of processing a sequence of colors arriving online using k servers and a buffer of size b. At any time step $t \geq 1$, a new item arrives and is immediately placed in the buffer. A server can move from color c to a different color c' at cost 1. He then processes all items of color c' currently in the buffer. As long as the server stays on a color, new items of that color are processed immediately without additional cost. The algorithm can only move to the next time step if the server contains at most b items. The goal is to minimize the total cost.

In order to simplify the presentation, we allow the algorithm to also perform *block operations* on the colors. A block operation on color c removes all items of that color from the buffer and incurs cost 1. Clearly, two server movements can simulate a block operation, so we changed the cost model outlined in the introduction by a constant factor only.

3.1.1. LP formulation

Our new linear programming relaxation for GRBM is shown below.

$$
\begin{array}{lrcll}
\min & \sum_{t,c} y_c(t) + \sum_{t,c} x_c(t) & & & \\
\text{s.t.} & \sum_c \delta_c(t) & \leq & k & \forall t \geq 1 \\
& \delta_c(t) & \leq & 1 & \forall c,\ t \geq 1 \\
& \delta_c(t-1) - \delta_c(t) & \leq & x_c(t) & \forall t \geq 1 \\
& \sum_c \big(\sum_{\tau \leq t} |E_c(\vec{v}, \tau)|_t \cdot y_c(\tau) + |E_c(\vec{v}, t)|_t \cdot \delta_c(t) \big) & \geq & |E(\vec{v}, t)| - b & \forall \vec{v},\ t \geq 1 \\
& y_c(t), x_c(t), \delta_c(t) & \geq & 0 &
\end{array}
\tag{$\mathrm{LP}_{k,b}$}
$$

For any time step t of the algorithm, variable $y_c(t) = 1$ if a block operation on color c is performed at time t. The variable $\delta_c(t) \leq 1$ indicates if a server is located at color c at the beginning of time step t. We set $\delta_c(0) = 0$ for all colors c. There exist k servers which gives constraint $\sum_c \delta_c(t) \leq k$. In order to account for server movements, we introduce variables $x_c(t)$. With the constraint $\delta_c(t-1) - \delta_c(t) \leq x_c(t)$ they indicate if a server moves away from color c at time t. The objective is to minimize the number of block operations $\sum_{t,c} y_c(t)$ plus the number of server movements $\sum_{t,c} x_c(t)$.

It remains to show how $\mathrm{LP}_{k,b}$ models the *buffer constraint* that a new time step only begins if at most b items are in the buffer. We first introduce some notation. For two vectors of time steps $\vec{v}, \vec{w} \in \mathbb{N}_0^n$ with $v_c \leq w_c$ for every color c, let $E_c(\vec{v}, \vec{w})$ be the set of items to color c arriving in the interval $[v_c, w_c)$. If $w_c = t$ for every color c, we simply write $E_c(\vec{v}, t)$; if $t \leq v_c$, the set $E_c(\vec{v}, t)$ is empty. The set of items arriving between time step vectors $\vec{v}$ and $\vec{w}$ is $E(\vec{v}, \vec{w}) = \bigcup_c E_c(\vec{v}, \vec{w})$; similarly, $E(\vec{v}, t) = \bigcup_c E_c(\vec{v}, t)$. In order to simplify the presentation, we set $|E_c(\vec{v}, t)| = 0$ if $t \leq v_c$.

The buffer constraint requires that the number of unprocessed items at the end of a time step is at most b. This is equivalent to requiring for each vector $\vec{v}$

$$
|E(\vec{v}, t)| - |\operatorname{buffer}(t)| \geq |E(\vec{v}, t)| - b \ , \tag{3.1}
$$

where $\operatorname{buffer}(t)$ is the current content of the buffer at the end of time step t. The left-hand side is at most the number of items that arrived since $\vec{v}$ and have already been removed. The number of items removed via block operations since $\vec{v}$ is at least $\sum_c \sum_{\tau \leq t} |E_c(\vec{v}, \tau)| y_c(\tau)$. For the colors that currently hold a server, the number of items that arrived since $\vec{v}$ is $\sum_c |E_c(\vec{v}, t)| \delta_c(t)$. This means that we can require the following

constraint:

$$\sum_{c} \sum_{\tau \leq t} |E_c(\vec{v}, \tau)| y_c(\tau) + \sum_{c} |E_c(\vec{v}, t)| \delta_c(t) \geq |E(\vec{v}, t)| - b \ . \tag{3.2}$$

Observe that the above constraint is not a faithful representation of the GRBM problem as an optimal solution might not satisfy it. This is because the constraint ignores items that have been removed by a server that has since changed location. We therefore require that the optimal solution accompanies every server movement from c to c' with a block operation on the old color c. This makes the optimal solution fulfill Constraint (3.2), and its cost increases by at most a factor of 2.

Simply using Constraint (3.2) for modeling the buffer constraint unfortunately leads to a large integrality gap of the LP. As an instructive example, consider a scenario with only block operations ($k = 0$) where a sequence of $b + 1$ items of the same color arrive. The optimal solution experiences at least cost 1 during the first $b + 1$ time steps, as it must perform a block operation. If the algorithm is, however, permitted to perform *fractional* block operations (via the $y_c(t)$-variables), it simply performs $1/b$ of a block operation in step $b + 1$ and proceeds to the next time step. This indicates an integrality gap of b, which is much more than we are aiming for.

The LP is therefore strengthened by adding so-called knapsack cover inequalities (see Carr et al. [CFLP00]). We proceed as follows. Whenever $|E_c(\vec{v}, \tau)| \geq |E(\vec{v}, t)| - b$, we decrease $|E_c(\vec{v}, \tau)|$ to $|E(\vec{v}, t)| - b$ in Constraint (3.2) . We claim that this does not change the feasibility of any integral solution. To see this, we note that this makes no change if $y_c(\tau)$ and $\delta_c(t)$ are 0. If $y_c(\tau) = 1$ or $\delta_c(\tau) = 1$, the left-hand side of the constraint is still at least $|E(\vec{v}, t)| - b$ afterwards; hence the constraint stays fulfilled. We define

$$|E_c(\vec{v}, \tau)|_t = \begin{cases} \min\{|E_c(\vec{v}, \tau)|, |E(\vec{v}, t)| - b\} & \text{if } |E(\vec{v}, t)| \geq b \\ 0 & \text{otherwise.} \end{cases}$$

Replacing $|E_c(\vec{v}, \tau)|$ by $|E_c(\vec{v}, \tau)|_t$ in Constraint (3.2) yields precisely the constraint used in the linear program $\mathrm{LP}_{k,\,b}$.

Linear program $\mathrm{LP}_{k,\,b}$ is in fact a generalized version of the one used by Adamaszek et al. for the block devices problem [ACER12]. To recover their formulation, set $k = 0$ and remove all constraints except the buffer constraint.

3.1.2. Modifying the buffer size

In the context of RBM, Englert and Westermann [EW05] first used the idea of comparing the performance of an online algorithm with the performance of an optimal algorithm with smaller buffer. A refined version of their argument is crucial in the analysis of RBM by Avigdor-Elgrabli and Rabani [AR13] and the analysis of the block-devices problem by Adamaszek et al. [ACER12]. In both problems, decreasing the buffer size by a factor of $\left(1 - \frac{1}{\log b}\right)$ only increases the cost by a constant factor. This section shows that the same holds for GRBM. This will allow us to formulate a stronger version of linear program $\mathrm{LP}_{k,b}$.

Theorem 3.1. *For any input sequence, the cost* $\mathrm{OPT}_{b'}$ *of an optimal offline solution utilizing a buffer of size* $b' = (1-\varepsilon)b > 0$ *is at most a factor of* $(3 + \varepsilon \ln b')/(1-\varepsilon)$ *larger than the cost* OPT_b *of an optimal offline solution utilizing a buffer of size* b*.*

Proof. This proof is a slightly adapted variant of proofs given by Englert et al. [ERW09] and Adamaszek et al. [ACER12].

Fix an input sequence and consider an optimal offline strategy using k servers and a buffer of size b. In a slight abuse of notation, we will call both the optimal strategy and its cost OPT_b. Without loss of generality, we may assume that OPT_b visits each color exactly once with a server; otherwise, we re-color remaining items of the color after a server first left it. This does not change OPT_b and can only increase OPT'_b. Also, without loss of generality, we assume OPT_b does not perform any block operations.

For a color c, let $t_{\mathrm{start}}(c)$ and $t_{\mathrm{end}}(c)$ denote the time step when a server of OPT_b visits and leaves c, respectively.[1] We call a color *finished* at time t if $t \geq t_{\mathrm{end}}(c)$. We call a color *active* if it is currently assigned a server.

In the following we construct a strategy for processing the input sequence with a buffer of size b'. The strategy performs the same server movements as the optimal strategy OPT_b and some additional block operations. Whenever the buffer is full and contains no active color, the strategy wants to perform the next server movement of OPT_b. This, however, may require it to move a server away from some color c before all items of that color have been processed, i.e., we still have $t < t_{\mathrm{end}}(c)$. In this case, the strategy defers the server movement and first performs block operations. These block operations remove additional items and thereby increase t to $t_{\mathrm{end}}(c)$ or beyond. At this point, the algorithm performs the previously deferred server movement.

[1] Here, we assume that only a single item is processed per time step, i.e., a block operation takes r time steps to process r items, but it does not process any new items of the same color arriving during that time. This clearly does not change our model.

When waiting for $t_{\text{end}}(c)$, the strategy performs block operations until a block operation is *successful*, i.e., if $t \geq t_{\text{end}}(c)$ holds afterwards. The strategy always chooses a color c' that causes the maximal total increase in *lateness* $\ell(c')$. Initially, $\ell(c') = 0$ for every color c. A successful block operation does not change the lateness of any color. If the block operation on c' is unsuccessful, the lateness $\ell(q)$ increases by $n(c')$ for every unfinished, inactive color q that finishes before $t_{\text{start}}(c')$ (including c' itself). Here, $n(c')$ denotes the number of items of color c' that are currently in the buffer.

Claim 3.2. *The lateness of a color is at most εb.*

Proof. Let c denote the inactive color that finishes next according to OPT_b. Among unfinished, inactive colors this color has the largest lateness $\ell(c)$. Assume we perform a block operation on some color q with $\ell(c) + n(q) > \varepsilon b$, i.e., the claim would be violated if the block operation were unsuccessful.

There at most b items in the input sequence that occur before $t_{\text{end}}(c)$ and that belong to colors q' with $t_{\text{end}}(c) < t_{\text{start}}(q')$, as otherwise OPT_b could not hold these items within its buffer. Observe that all items that contributed to the increase of $\ell(c)$ and also the $n(q)$ items of q belong to this class of items. Since the strategy for buffer size b' has removed them, there are at most $b - \ell(c) - n(q) < b - \varepsilon b = b'$ of these items left. This means that the strategy can hold all of these items in its buffer and can therefore advance to $t_{\text{end}}(c)$. Hence, the block operation on q is successful, which is a contradiction. □

Claim 3.3. *There exists a color c' such that $\sum_q \Delta\ell(q) \geq b'/(1 + \ln b')$, where $\Delta\ell(q)$ is the increase of the lateness of color q in case the block operation on c' is unsuccessful.*

Proof. Let $s = b'/(1 + \ln b')$ to shorten notation, and assume for contradiction that the increase in total lateness for every color is less than s.

Every server movement finishes one color and starts a new one. Sort the inactive unfinished colors according to their start times and let c_i be the i-th color among them. If follows that there are i currently unfinished colors that end before $t_{\text{start}}(c_i)$. Performing a block operation on c_i incurs an increase of at least $i \cdot n(c_i)$ in total lateness. As the total lateness increase for every color is assumed to be at most s, it follows that $n(c_i) = 0$ for $i > s$. Also note that active colors and finished colors have no items in the buffer. Since the buffer is full, it follows that

$$b' = \sum_{i=1}^{\lfloor s \rfloor} n(c_i) \leq s \sum_{i=1}^{\lfloor s \rfloor} \frac{1}{i} < s(1 + \ln s) = \frac{b'}{1 + \ln b'}(1 + \ln s) < b' \ ,$$

which is a contradiction. □

Our strategy always performs a block operation on a color with maximal increase in lateness, as described in Claim 3.3. Every successful block operation is followed by a server movement that is also performed by OPT_b, thus there can be at most OPT_b successful block operations. Every unsuccessful block operation increases $\sum_c \ell(c)$ by at least $b'/(1+\ln b')$ according to Claim 3.3. The total lateness amassed throughout the algorithm is at most $\varepsilon b\,\mathrm{OPT}_b$ by Claim 3.2. Hence, at most $\varepsilon b(1+\ln b')/b'\,\mathrm{OPT}_b = \frac{\varepsilon}{1-\varepsilon}(1+\ln b')\,\mathrm{OPT}_b$ unsuccessful block operations are performed by the strategy. Finally, the strategy performs OPT_b server movements imitating OPT_b. Its total cost is therefore

$$(2+\frac{\varepsilon}{1-\varepsilon}(1+\ln b'))\,\mathrm{OPT}_b < (3+\ln b')/(1-\varepsilon)\cdot\mathrm{OPT}_b \quad . \qquad \square$$

Using the above theorem, we perform the following modifications to our linear program. Let $\varepsilon = 1/\ln b$, so $b' = (1-1/\ln b)b$. Then the optimal solution value of $\mathrm{LP}_{k,b}$ is at most a constant factor smaller than the one of $\mathrm{LP}_{k,b'}$. We furthermore remove all buffer constraints for pairs $(\vec{v},t)$ with $|E(\vec{v},t)| \le b$ from $\mathrm{LP}_{k,b'}$, which does not increase the optimum solution value. The resulting LP Primal and its dual program shown below.

$$\begin{array}{lrcll}
\min & \sum_{t,c} y_c(t) + \sum_{t,c} x_c(t) & & & \\
\text{s.t.} & \sum_c \delta_c(t) & \le & k & Z(t) \\
& \delta_c(t) & \le & 1 & p_c(t) \\
& \delta_c(t-1) - \delta_c(t) & \le & x_c(t) & B_c(t-1) \\
\multicolumn{5}{c}{\forall\ \vec{v},t \text{ with } |E(\vec{v},t)| \ge b:} \\
\sum_c\big(\sum_{\tau\le t} |E_c(\vec{v},\tau)|_t y_c(\tau) + |E_c(\vec{v},t)|_t \delta_c(t)\big) & & \ge & |E(\vec{v},t)| - b' & \alpha(\vec{v},t) \\
& y_c(t), x_c(t), \delta_c(t) & \ge & 0 &
\end{array} \qquad \text{(Primal)}$$

$$\begin{array}{lrcll}
\max & \sum_{\vec{v},t}\big(|E(\vec{v},t)| - b'\big)\alpha(\vec{v},t) - k\sum_t Z(t) - \sum_{c,t} p_c(t) & & & \\
\text{s.t.} & \sum_{\vec{v},t\ge\tau} |E_c(\vec{v},\tau)|_t \cdot \alpha(\vec{v},t) & \le & 1 & y_c(\tau) \\
& B_c(t-1) & \le & 1 & x_c(t) \\
-p_c(t) - Z(t) - B_c(t) + B_c(t-1) + \sum_{\vec{v}} |E_c(\vec{v},t)|_t \cdot \alpha(\vec{v},t) & & \le & 0 & \delta_c(t) \\
& \alpha(\vec{v},t), p_c(t), B_c(t), Z(t) & \ge & 0 &
\end{array} \qquad \text{(Dual)}$$

We obtain the following corollary.

Corollary 3.4 ([ACER12]). *For $b' = \left(1 - \frac{1}{\log b}\right)b$, the value of LP Primal is at most a constant factor larger than the cost OPT_b of an optimal offline algorithm using a buffer of size b.*

Proof. Follows from Theorem 3.1 and the above discussion. □

3.2. Algorithm Overview

Our algorithm and its analysis are divided into two main parts, the *base procedure* and the *cost control procedure* wrapped around it. The base procedure is essentially the $\mathcal{O}(\log \log b)$-competitive algorithm for scheduling block-devices by Adamaszek et al. [ACER12]. This algorithm only schedules block operations but it never moves a server. If left alone, it produces a valid solution to GRBM, but this solution is not competitive with an optimal solution. The cost control procedure therefore schedules additional block operations and it specifies a movement strategy for the k servers with the goal to greatly reduce the number of block operations performed by the base procedure.

The base procedure is an LP-based algorithm using the primal-dual schema of Buchbinder and Naor [BN09]. At each time step, the algorithm picks a violated primal constraint $(\vec{v}, t)$ and increases the corresponding dual variable $\alpha(\vec{s}, t)$ continuously. The resulting violation of the dual constraints then triggers a fractional increase of primal variables $y_c(\tau)$. In order to obtain an integral solution that can be translated into actual block operations, the base procedure applies a randomized rounding scheme. The randomized rounding makes sure that the buffer constraint is fulfilled, i.e., it ensures that at most b items reside in the buffer before the next time step. In the analysis, the cost the solution is compared against the profit of the dual solution the algorithm constructed.

The cost control procedure is built around the base procedure in order to adapt it to GRBM. When a new item arrives in a buffer of b items, the cost control procedure may accommodate the new item by performing a block operation itself. Alternatively, it can choose to leave the item to the base procedure. In the latter case, the cost control procedure may react to the assignment of primal and dual variables by the base procedure.

The cost control procedure is vaguely inspired by the randomized *Marking* algorithm for the paging problem by Fiat et al. [FKL$^+$91]. In contrast to that algorithm, however, the cost control procedure is nontrivial even in the case of a single server. We note that the cost control procedure is deterministic for $k = 1$ and it only performs integral assignments to the LP variables. For $k > 1$, the procedure is randomized, yet it does not employ a randomized rounding procedure.

Purely to simplify the presentation of our algorithm and our proofs we introduce dummy time steps. These are time steps in which no new item arrives. We assume that between any two real time steps (in which an item arrives) there are an arbitrarily large number of dummy time steps. This allows us to present our algorithm as a nearly continuous process, whenever this is convenient.

We now describe the algorithm's main components in more detail.

3.2.1. Base procedure

The base procedure works on the $\alpha(\vec{z}, t)$-constraints of the linear program Primal and its dual. Given a (partial) assignment to the variables of the primal and a time step t, we introduce the notion of a *proper constraint*. The definition uses a constant scaling factor $\beta := 80$.

Definition 3.5. For a given variable assignment, an $\alpha(\vec{z}, t)$ primal constraint is called *proper* if the following holds for every color c:

(i) $\sum_{z_c \leq \tau \leq t}(y_c(\tau) + \delta_c(\tau)) \geq 1/\beta$,

(ii) $\sum_{z_c < \tau \leq t}(y_c(\tau) + \delta_c(\tau)) < 2/\beta$,

(iii) $y_c(\tau) + \delta_c(\tau) < 1/\beta$ for all $\tau \in \{z_c + 1, \ldots, t\}$.

Intuitively, a proper constraint sets z_c to the time of the last block operation. In particular, whenever there is an integral server movement on color c (so $\delta_c(\tau) = 1$ or $y_c(\tau) = 1$), then for a proper constraint $\alpha(\vec{z}, t)$ we must have $z_c \geq \tau$.

The main loop of our algorithm feeds proper constraints $\alpha(\vec{z}, t)$ to the base procedure. The base procedure then increases the $y_c(\tau)$-variables of the constraint in typical primal-dual fashion, as shown in Algorithm 1. The procedure terminates if either (1) the current constraint is fulfilled, or (2) the current constraint ceases to be a proper constraint.

If the current constraint is fulfilled, a rounding algorithm takes over in Line 14. It uses the fractional assignment to the $y_c(\tau)$-variables and produces a probabilistic distribution μ over block operation schedules. Each of the schedules in this distribution respects the buffer constraint. Importantly, the rounding procedure does not require that the fractional assignment fulfills every $\alpha(\vec{v}, t)$-constraint. We defer the description of the rounding algorithm to Section 3.3.2.

In the second case, the base procedure notices that the current constraint is not proper anymore (Line 1). It then recomputes the vectors $\vec{s}$ and $\vec{z}$ (Line 2). We now describe these two vectors in more detail. The cost control procedure keeps vectors $\vec{s}$ and $\vec{z}$ along

Algorithm 1: Base procedure

if *primal constraint* $\alpha(\vec{z},t)$ *is not proper* **then**
  recompute $\vec{z}$ and $\vec{s}$
if *primal constraint* $\alpha(\vec{z},t)$ *is violated* **then**
  Increase $\alpha_1(\vec{z},t)$ and $\alpha_2(\vec{s},t)$ by the same infinitesimal amount $\mathrm{d}\alpha$
  foreach *variable* $y_c(\tau)$, $z_c < \tau \leq t$ **do**
    $\Delta(\tau,c) := 0$
    if $\sum_{\vec{v},t' \geq \tau} |E_c(\vec{v},\tau)|_{t'} \cdot \alpha_1(\vec{v},t') == 1$ **then**
      // dual constraint $y_c(\tau)$ tight
      if $(\sum_{\tau \leq i \leq t} y_c(i) < \frac{1}{\log^3 b})$ **then**
        $\hat{y}_c(t) := \frac{1}{\log^3 b}$
    if $\sum_{\vec{v},t' \geq \tau} |E_c(\vec{v},\tau)|_{t'} \cdot \alpha_1(\vec{v},t') > 1$ **then**
      // dual constraint $y_c(\tau)$ violated
      $\Delta(\tau,c) := \sum_{\tau \leq i \leq t} y_c(i) \cdot |E_c(\vec{z},\tau)|_t \cdot \mathrm{d}\alpha$
  $\forall c : \mathrm{d}y_c(t) := \hat{y}_c(t) + \max_{\tau : z_c < \tau \leq t} \Delta(\tau,c)$
  $\forall c : y_c(t) := y_c(t) + \mathrm{d}y_c(t)$
  adjust distribution μ to reflect new y-values
else // proper primal constraint $\alpha(\vec{z},t)$ is satisfied
   // rounding guarantees at most b items stored in buffer
  proceed to the next time step

with corresponding (sets of) variables $\alpha_1(\vec{z},t)$ and $\alpha_2(\vec{s},t)$. The base procedure increases both variables but it only bases its decisions on the α_1-variables; the α_2-variables are discussed in the next section. For vector $\vec{z}$, the rounding procedure only requires that $\alpha(\vec{z},t)$ be a proper constraint and that the vectors $\vec{z}$ be monotonously increasing over time. Hence, for computing a new vector $\vec{z}$, every coordinate z_c that violates the constraints of Definition 3.5 is set to the largest value less or equal to the current time t so that $\sum_{v_c \leq \tau \leq t}(y_c(\tau) + \delta_c(\tau)) \geq 1/\beta$. Observe that we may assume without loss of generality that such a value always exists if we perform a block operation on any color that appears for the first time[2]. The vector $\vec{s}$ is managed by the cost control procedure. For any color c, $s_c = z_c$ by default, but the algorithm may decide to set $s_c < z_c$ for some colors. We postpone the detailed description to the next section.

[2]This change increases the algorithm's cost by at most a factor of 2

3.2.2. Cost control procedure

The cost control procedure essentially consists of two new subroutines that wrap around the base procedure. The *activation procedure* reacts when new items arrive to "activate" colors and perform block operations. The *deactivation procedure* reacts when the base procedure increases primal and dual variables of the LP. The procedure "deactivates" colors and it is responsible for moving the k servers. Both procedures integrally set variables of LP Primal and control the vector $\vec{s}$. By setting variables of the LP, the cost control procedure may change the current proper constraint $\alpha(\vec{z}, t)$. This is how the base procedure indirectly learns about the decisions of the cost control procedure. We next define the concept of activation and deactivation in more detail.

The interplay between activation procedure and deactivation procedure is governed by the notion of an *active color*. Very informally, a color is *active*, if it has seen a significant number of items since it was last visited by a server or has experienced a block operation. Our goal is that the active colors contribute a large profit via the dual LP. A color is *deactivated*, i.e., it ceases to be active, when it has caused too much violation in the dual program.

More precisely, recall that the base procedure uses two vectors $\vec{s}$ and $\vec{z}$ and it requires that $\alpha(\vec{z}, t)$ is always a proper constraint. For inactive colors, $z_c = s_c$ and all colors start out as inactive. When color c becomes active, the procedure sets $s_c < z_c$ and "freezes" s_c, i.e., s_c remains fixed until c is deactivated. The value of s_c is chosen so that $|E_c(\vec{s}, t)| = \lceil f_{\max} \rceil$, with

$$f_{\max} = \frac{b - b'}{2k} = \frac{b}{2k \log b} \ .$$

We distinguish the set of *live items* $E_c(\vec{z}, t)$ of a color c and the set of *extra items* $E_c(\vec{s}, \vec{z})$. Observe that $s_c = z_c$ for inactive colors, hence the set of extra items inactive colors. The rounding routine of the base procedure ensures that all but the live items have been removed from the buffer. The profit generated by the extra items (through $\alpha_2(\vec{s}, t)$-variables in the base procedure) is used to pay for the actions of the cost control procedure. We show below that the activation procedure makes sure that the number of extra items is at least a constant fraction of the number of live items for each active color. Before doing so, we explain the activation procedure.

The activation procedure

The activation procedure is shown in Algorithm 2. The procedure activates colors and generates a sequence of *block requests* for the active colors. If the algorithm does not

Algorithm 2: Activation procedure

```
   // Item of color c arrives to a buffer of b items
   // Block-Request(c) performs block operation on c unless c has server
1  if Color c is inactive and unmarked then
2      if Less than f_max items for c arrived since last block request then
3          Base-Procedure(t)
4      else
5          Color c becomes active, set s_c < z_c
6          Block-Request(c)
7  else // Color c is active
8      if |E_c(s⃗, t)| < (1 + γ)|E_c(s⃗, t')| then // t' is time of last block req.
9          Base-Procedure(t)
10     else if at least D block requests to c during current act. period then
11         Base-Procedure(t)
12     else
13         Block-Request(c)
```

have a server at color c, it must react to a block request to c with an immediate block operation on c. If a server is located at c, no action is necessary. The rules for activating colors use the notion of a color being either *marked* or *unmarked*. Additionally, unmarked colors can be *stale*. These terms are borrowed from paging algorithms and we will only define them later precisely. For now, we only need that the activation procedure makes sure the following constraint is always observed.

scheduling constraint:

- a marked color is always assigned a server,
- the remaining colors are only distributed among stale colors.

What follows are the rules for generating block requests.

- If a color is not active and *unmarked*, the activation procedure issues a block request after $\lceil f_{\max} \rceil$ items arrived since the last block request to the color. With this block request, the color becomes active.
- For an active color c, let t' be the time of the last block request to c. The color obtains a block request when $|E_c(\vec{s}, t)| \geq (1+\gamma)|E_c(\vec{s}, t')|$, which means that a new

block request is issued after $\gamma|E_c(\vec{s},t')|$ new items arrived. We use the constant $\gamma = 1/128$.

- As an exception to the previous rule, no further requests for a color are issued once it has obtained $D := \lceil \log_{1+\gamma}((2b+2)/(\gamma f_{\max}))\rceil + 1 = \Theta(\log k + \log\log b)$ requests within its current activity period.

Observe that after a block request to c, $z_c \geq t$. This is because after a block request either $\delta_c(t) = 1$ or $y_c(t) = 1$; by Condition (iii) of the proper constraints, z_c must increase in order for $\alpha(\vec{z},t)$ to stay proper.

We now show that for active colors, nearly all items in the set $E_c(\vec{s},t)$ are extra items.

Lemma 3.6. *For active colors, the number of live items is at most a γ-fraction of the number of extra items, more precisely,*

$$|E_c(\vec{z},t)| \leq \gamma|E_c(\vec{s},\vec{z})| \ .$$

Proof. We first show $|E_c(\vec{z},t)| \leq 2b$. Consider the $\alpha(\vec{z},t)$-constraint in LP Primal

$$\sum_c \Big(\sum_{\tau\leq t} |E_c(\vec{z},\tau)|_t \cdot y_c(\tau) + |E_c(\vec{z},t)|_t \cdot \delta_c(t)\Big) \geq |E(\vec{z},t)| - b \ .$$

As $|E_c(\vec{z},\tau)|_t \leq |E_c(\vec{z},t)|$, the left-hand side is at most

$$\sum_c |E(\vec{z},t)| \sum_{\tau=z_c+1}^{t} (y_c(\tau) + \delta_c(t)) \leq \frac{2}{\beta}|E(\vec{z},t)| \ . \tag{3.3}$$

The inequality follows from Condition (ii) for proper constraints.

On the other hand, the $\alpha(\vec{z},t)$ constraint is violated by at most an additive 1. To see this, observe that we can think of any increase of primal variables as infinitesimal, even if performed outside the base procedure. In this scenario, whenever the current constraint is fulfilled, the algorithm chooses a new proper constraint. Hence, the algorithm ends a time step with a proper, fulfilled constraint. Moving to a new time step (with the same vector $\vec{z}$) can only generate violation 1. This implies that the left hand side of the $\alpha(\vec{z},t)$-constraint is at least $|E(\vec{z},t)| - b - 1$. In combination with the upper bound (3.3) on the left-hand side, it follows that

$$|E(\vec{z},t)| - b - 1 \leq \frac{2}{\beta}|E(\vec{z},t)| \ .$$

Algorithm 3: Deactivation procedure

```
1  if volume vol_c^T((t_c^act, t]) ≥ V for some active color c then
2      Label c as ready for deactivation
3  if (
4      volume vol^T((t_c^act, t]) ≥ 3V for some active color c or
       a block operation was performed color c ready for deactivation
5  ) then
6      Deactivate c and set s_c = z_c
7      if No server is located at c then
8          Perform a block operation on c
9      if activation period of c started in the current phase then
10         Move a server to c and mark c
11         if k colors have been marked in the current phase then
12             End the current phase and start a new one
```

As $\beta = 80 > 4$, this implies

$$|E(\vec{z}, t)| \leq \frac{\beta}{\beta - 2}(b+1) < 2b + 1 \ ,$$

so we obtain the upper bound in b.

Now we show $|E_c(\vec{z}, t)| \leq \gamma |E_c(\vec{s}, \vec{z})|$. Consider the case that color c has seen less than D requests in the current activity period. Assume for contradiction that $|E_c(\vec{z}, t)| > \gamma |E_c(\vec{s}, \vec{z})|$, then

$$|E_c(\vec{s}, t)| = |E_c(\vec{s}, \vec{z})| + |E_c(\vec{z}, t)| > (1+\gamma)|E_c(\vec{s}, \vec{z})| \geq (1+\gamma)|E_c(\vec{s}, t')| \ ,$$

which triggers a block request for color c. The last inequality follows as $z_c \geq t'$ after a block request to c. Hence, the block request restores the claimed property.

Now suppose color c has seen D block requests during the current activity period and recall that there can be no more than D requests per color. Right after the i-th request we have $|E_c(\vec{s}, t)| = |E_c(\vec{s}, \vec{z})| \geq (1+\gamma)^{i-1} f_{\max}$. It follows that $|E_c(\vec{s}, \vec{z})|$ is at least $(1+\gamma)^{D-1} \geq \frac{2b+2}{\gamma}$. In order to violate the claim we would therefore require $|E_c(\vec{z}, t)| \geq 2b + 2$, which is impossible. □

The deactivation procedure

The deactivation procedure is given in Algorithm 3. The deactivation procedure reacts to the increase of the dual variables α_2 by the base procedure. Its first goal is to bound the violation of the dual program through $\alpha_2(\vec{s}, t)$-variables in the base procedure. Its second goal is limit the cost incurred by the block requests of the activation procedure. The deactivation procedure deactivates colors that contributed too much violation and it may then decide to mark that color. Recall that, the scheduling constraint states that a marked color always has a server assigned to it; hence, the deactivation procedure also controls the server movements.

We now introduce the notion of *total volume* that is central to the deactivation procedure. For a time interval I, let the total volume $\mathrm{vol}_c^T(I)$ be the total *increase* of the sum $\sum_{v,\tau \leq t} |E(\vec{v}, \tau)| \alpha(\vec{v}, \tau)$ during interval I. For an active color c, let t_c^{act} be the time c was activated, so $\mathrm{vol}_c^T((t_c^{\mathrm{act}}, t])$ is the volume collected by c since its activation. Observe that this is precisely the increase of the left-hand side of the $y_c(t)$-constraint. We will formalize this idea in the next section.

Ideally, we want to deactivate a color as soon as it has collected volume V since its activation, where $V = \mathcal{O}(\log k + \log \log b)$ will be specified later. For technical reasons, we instead set a color to be *ready for deactivation* when it has collected volume V. If a block operation is performed on a color that is ready, we immediately deactivate it. Only if the volume collected by the color reaches $3V$ we *force* a deactivation and deactivate the color immediately. This rule permits us to amortize, in most cases, the cost of deactivation against the cost of the block operation.

When a color c is deactivated, the procedure performs a block operation on c, unless a server resides there. The procedure also decides whether to mark it or not. This marking process is similar to the *Marking* algorithm. The algorithm partitions the time horizon into *phases*. A phase starts with all colors unmarked. A a color is marked, if its activation and deactivation lie in the same phase. When k colors are marked, all colors are unmarked and a new phase starts. The choice, which of the k servers to move to a newly marked color induces a variant of the paging problem. We defer the description and analysis of this subroutine to Section 3.4.

3.3. Analysis

The analysis of the algorithm defined in the previous section is guided by the value of the linear programs Primal and Dual. We see in this section that the expected cost of the algorithm is proportional to the final objective value of the primal, i.e.,

$\mathcal{O}\big(\sum_{t,c} y_c(t) + \sum_{t,c} x_c(t)\big)$. Furthermore, the base procedure increases the α_1- and α_2-variables of the dual, which gives a partial solution to this linear program. By weak duality, the value of the dual is a lower bound on the value of the primal and, hence, the cost of an optimal solution. Our high-level strategy is therefore to extend the assignment to primal variables from the dual so that we obtain a lower bound that pays for both the base procedure and the cost control procedure.

It is convenient to view its decisions as a continuous process of the α-variables. Recall that the base procedure increases a single $\alpha_1(\vec{s}, t)$ and a single $\alpha_2(\vec{z}, t)$ variable in each time step at the same infinitesimal rate $\mathrm{d}\alpha$. Let α denote the sum of α_2-variables (or α_1-variables) and observe that α is nondecreasing over time. We can therefore view $\vec{s}$, $\vec{z}$ and t as functions of α (instead of functions of time) and we write $\vec{s}(\alpha)$, $\vec{z}(\alpha)$ and $t(\alpha)$. The total profit contributed by only the α_2-variables is then

$$\int_0^{\alpha_{\max}} (|E(\vec{s}(\alpha), t(\alpha)|) - b') \,\mathrm{d}\alpha - k \sum_t Z(t) - \sum_{c,t} p_c(t) \ , \tag{3.4}$$

where $\alpha_{\max}$ is the sum of all α_2-variables in the end. In order to avoid notational clutter we write $|E(\vec{s}, t)|$ instead of $|E(\vec{s}(\alpha), t(\alpha)|)$ whenever the dependency on α is clear. Recall that $\vec{s} \leq \vec{z}$ at all times, so $|E(\vec{s}, t)| \geq |E(\vec{z}, t)|$. This means that the α_1-variable generate a smaller profit than the α_2-variables and we ignore the former for our lower bound construction.

The central idea of our algorithm is separating it into two parts: the base procedure on the one hand and the cost control procedure on the other hand. Analogously, we split the profit (3.4) in a *base budget* $\mathcal{B}_{\text{base}}$ and a *cost control budget* $\mathcal{B}_{\text{control}}$ as follows:

$$\begin{aligned}
\mathcal{B}_{\text{base}} &= \frac{1}{2} \int_0^{\alpha_{\max}} (|E(\vec{z}, t)| - b') \,\mathrm{d}\alpha \ , \\
\mathcal{B}_{\text{control}} &= \frac{1}{2} \int_0^{\alpha_{\max}} (|E(\vec{z}, t)| - b') \,\mathrm{d}\alpha + \int_0^{\alpha_{\max}} |E(\vec{s}, \vec{z})| \,\mathrm{d}\alpha - k \sum_t Z(t) - \sum_{c,t} p_c(t) \ .
\end{aligned}$$

We first show how to set the $Z(t)$-, $p_c(t)$- and $B_c(t)$-variables of the dual program so that we obtain a large cost control budget with small violation of the dual. In particular, it is important that we assign values to dual variables such that the control budget $\mathcal{B}_{\text{control}}$ is positive. We therefore define a set of conditions for the cost control procedure to fulfill.

Second, we show that the cost of the base procedure (Procedure 1) is at most $\mathcal{B}_{\text{base}}$. Recall that the base procedure is a primal-dual algorithm that schedules block operations on the colors. It produces a *fractional* assignment to the $y_c(t)$-variables. A randomized rounding procedure generates a probability distribution over valid schedules for block

operations based on the fractional assignment. The analysis of the base procedure is straightforward, as it is essentially the algorithm by Adamaszek et al. for scheduling block-devices [ACER12]. The authors show that the increase in $y_c(t)$-variables per iteration is at most $2\sum_{\vec{v},t}\big(|E(\vec{v},t)| - b'\big)\alpha_1(\vec{v},t)$. This means, up to a factor of 4, it is bounded by the base budget $\mathcal{B}_{\text{base}}$. Furthermore, the expected cost of the schedules produced by the randomized rounding procedure is at most $\mathcal{O}\big(\sum_{t,c} y_c(t)\big)$. We include their analysis for completeness in Section 3.3.2.

Section 3.3.3 then shows that our cost control procedure indeed fulfills the constraints that guarantee a large cost control budget. This leads to a proof of the competitive ratio $\mathcal{O}(\log\log b)$ for the case of $k = 1$. Furthermore, we obtain a set of conditions for the subroutine for scheduling $k > 1$ servers. In Section 3.4, an algorithm fulfilling these constraints is presented, which implies an online algorithm for GRBM with polylogarithmic competitive ratio.

3.3.1. Lower bound construction

We now develop a lower bound on the cost on the optimal solution. We start with some more notation. Let for a subset $I \subseteq \{1,\dots,T\}$ of (consecutive) time steps $\alpha^-(I) := \inf\{\alpha \mid t(\alpha) \in I\}$ and $\alpha^+(I) := \sup\{\alpha \mid t(\alpha) \in I\}$. The profit collected for the cost control budget during some time interval I is

$$\begin{aligned}\mathcal{B}_{\text{control}}(I) &= \frac{1}{2}\int_{\alpha^-(I)}^{\alpha^+(I)} (|E(\vec{z},t)| - b')\,\mathrm{d}\alpha + \int_{\alpha^-(I)}^{\alpha^+(I)} |E(\vec{s},\vec{z})|\,\mathrm{d}\alpha - k\sum_{t\in I} Z(t) - \sum_c \sum_{t\in I} p_c(t)\\ &\geq \int_{\alpha^-(I)}^{\alpha^+(I)} |E(\vec{s},\vec{z})|\,\mathrm{d}\alpha - k\sum_{t\in I} Z(t) - \sum_c \sum_{t\in I} p_c(t) + k(\alpha^+(I) - \alpha^-(I))f_{\max}\ .\end{aligned}$$

The inequality follows as $|E(\vec{v},t)| - b' \geq b/\log b$ for any $\alpha(\vec{v},t)$ constraint in LP Primal, so with $f_{\max} = b/(2k\log b)$

$$\frac{1}{2}\int_{\alpha^-(I)}^{\alpha^+(I)} (|E(\vec{z},t)| - b')\,\mathrm{d}\alpha \geq (\alpha^+(I) - \alpha^-(I))\frac{b}{2\log b} = k(\alpha^+(I) - \alpha^-(I))f_{\max}\ .$$

We use $\text{act}_c(I)$ and $\overline{\text{act}}_c(I)$ to denote the subset of the interval $[\alpha^-(I), \alpha^+(I)]$ during which color c is active and inactive, respectively. We define the *extra volume* collected during interval I for color c by

$$\text{vol}_c^E(I) = \int_{\alpha^-(I)}^{\alpha^+(I)} |E_c(\vec{s},\vec{z})|\,\mathrm{d}\alpha = \int_{\text{act}_c(I)} |E_c(\vec{s},\vec{z})|\,\mathrm{d}\alpha\ .$$

The equality follows because s_c and z_c are equal whenever c is not active. We define the *total volume* of a color c collected during interval I by

$$\mathrm{vol}_c^T(I) = \int_{\mathrm{act}_c(I)} |E_c(\vec{s},t)|\,\mathrm{d}\alpha \ .$$

Summing $\mathrm{vol}_c^T(I)$ over all colors, we obtain the increase of $\sum_{v,\tau\le t}|E(\vec{v},\tau)|\alpha(\vec{v},\tau)$ during interval I. The definition of total volume of a color c is therefore consistent with the definition of total volume of the previous section.

We also define the *gap* $\Delta_c(I)$ between total volume and extra volume for a time interval I:

$$\Delta_c(I) := \mathrm{vol}_c^T(I) - \mathrm{vol}_c^E(I) \le \gamma \mathrm{vol}_c^E(I) \ . \tag{3.5}$$

The inequality follows because our algorithm ensures that an active color has $|E_c(\vec{s},t)| \le (1+\gamma)|E_c(\vec{s},\vec{z})|$ (Lemma 3.6). We extend the above definitions to individual time steps, i.e., we define $\mathrm{vol}_c^T(t) := \mathrm{vol}_c^T(\{t\})$. Observe that this means that $\mathrm{vol}_c^T(I) = \sum_{t\in I}\mathrm{vol}_c^T(t)$. This allows us to rewrite the cost control budget for a time interval as

$$\mathcal{B}_{\mathrm{control}}(I) \ge \sum_{t\in I}\sum_c \mathrm{vol}_c^E(t) - \sum_{t\in I}\sum_c p_c(t) - k\sum_{t\in I} Z(t) + k(\alpha^+(I)-\alpha^-(I))f_{\max} \ . \tag{3.6}$$

We use $X_c(t) := \sum_v |E_c(\vec{v},t)|_t\alpha(\vec{v},t)$, which is the term that appears in the $\delta_c(t)$-constraint of the dual. We have

$$\begin{aligned} X_c(t) &= \int_{\alpha^-(t)}^{\alpha^+(t)} |E_c(\vec{s},t)|_t\,\mathrm{d}\alpha \le \int_{\alpha^-(t)}^{\alpha^+(t)} |E_c(\vec{s},t)|\,\mathrm{d}\alpha \\ &= \int_{\mathrm{act}_c(t)} |E_c(\vec{s},t)|\,\mathrm{d}\alpha + \int_{\overline{\mathrm{act}}_c(t)} |E_c(\vec{s},t)|\,\mathrm{d}\alpha \\ &\le \mathrm{vol}_c^T(t) + (\alpha^+(I)-\alpha^-(I))f_{\max} \ . \end{aligned}$$

The second inequality follows as an inactive color has $s_c = z_c$ and has seen less than $f_{\max}$ items since it became inactive. This gives that the $\delta_c(t)$-constraint of the dual is

$$X_c(t) - p_c(t) \le Z(t) + B_c(t) - B_c(t-1) \ ,$$

which is implied by

$$(\alpha^+(I)-\alpha^-(I))f_{\max} + \mathrm{vol}_c^T(t) - p_c(t) \le Z(t) + B_c(t) - B_c(t-1) \ . \tag{3.7}$$

We now present the lower bound construction. Suppose we are given a partial solution

to LP Dual that only specifies values for α-variables but that ensures that the $y_c(\tau)$-constraints are violated by at most a factor of W. We describe how to extend this solution in order to get a dual solution with large profit and, hence, a large lower bound for the problem.

Recall that the marking process of our algorithm partitions the time horizon into *phases*. We now specify three constraints for the phases that imply a large profit. In later sections, we show that these constraints are actually respected by our algorithm.

Consider some partitioning of the time horizon into phases. We say that the i-th phase consists of time steps $I_i := \{\tau_i, \ldots, \tau_{i+1} - 1\}$ with $\tau_1 = 1$. We require that each phase contain exactly k marked colors. We use M_i to denote the set of marked colors in the phase and U_i to denote the set of the remaining (unmarked) colors. Let V be a parameter to be defined later. We impose the following three constraints on the phases.

Marked Colors Each marked color c collects volume at least V during the phase, i.e., for each $c \in M_i$

$$\mathrm{vol}_c^T(I_i) = \int_{\mathrm{act}_c(I_i)} |E_c(\vec{s}, t)| \, \mathrm{d}\alpha \geq V \ . \qquad \text{(Condition I)}$$

Unmarked Colors Each unmarked color c collects volume at most $6V$ during a phase, i.e., for each $c \in U_i$

$$\mathrm{vol}_c^T(I_i) = \int_{\mathrm{act}_c(I_i)} |E_c(\vec{s}, t)| \, \mathrm{d}\alpha \leq 6V \ . \qquad \text{(Condition II)}$$

New Colors We call a marked color *new* if it was unmarked in the previous phase. Let n_i denote the number of new colors in the i-th phase. The gap between total volume and extra volume is small for marked colors, i.e.,

$$\sum_{c \in M_i} \Delta_c(I_i) \leq 10\gamma n_i V \ . \qquad \text{(Condition III)}$$

This is sufficient to obtain our main lower bound.

Lemma 3.7. *Suppose we are given an assignment to α-variables that violates $y_c(\tau)$-constraints by at most a factor of $W \geq V$, together with a partitioning of time steps into phases according to the above constraints. Then there exists a constant $\xi > 0$ such that*

the cost of an optimum solution is at least

$$\frac{\xi}{W}\left(\sum_i n_i V + \sum_i \sum_{c\in U_i} \mathrm{vol}_c^T(I_i)\right) + \frac{1}{2W}\int_0^{\alpha_{\max}} \Big(|E(\vec{z},t)| - b'\Big)\,\mathrm{d}\alpha \ .$$

Proof. Fix a pair of consecutive phases i and $i+1$, and let $I := \{\tau_i, \ldots, \tau_{i+2}-1\}$ denote the time steps within these phases. In the following we define values for the LP variables apart from $\alpha(\vec{s},t)$ in order to obtain a solution to LP Dual.

Our first step is to use the $p_c(t)$-variables to *normalize* the volume. We define $\mathrm{vol}_c^N(t) = \mathrm{vol}_c^T(t) - p_c(t)$, and call $\mathrm{vol}_c^N(t)$ the *normalized* volume of color c at step t. We require that $\mathrm{vol}_c^N(I) = \min\{\mathrm{vol}_c^T(I), V\}$ for each color and we set the $p_c(t)$-variables accordingly. Since this means we only have to "reduce" volume we can achieve this with a non-negative assignment to the $p_c(t)$-variables. Observe that Condition I implies any color in $M_i \cup M_{i+1}$ will have $\mathrm{vol}_c^N(I) = V$, as its total volume in one of the phases must be at least V.

Rewriting Equation (3.6) gives a cost control budget for interval I of

$$\begin{aligned}\mathcal{B}_{\mathrm{control}}(I) &= \sum_{t\in I}\sum_c \mathrm{vol}_c^E(t) - \sum_{t\in I}\sum_c p_c(t) - k\sum_{t\in I} Z(t) + k(\alpha^+(I) - \alpha^-(I))f_{\max} \\ &= \sum_{t\in I}\sum_c \Big(\mathrm{vol}_c^N(t) - \Delta_c(t)\Big) - k\sum_{t\in I} Z(t) + k(\alpha^+(I) - \alpha^-(I))f_{\max} \ .\end{aligned}$$

We now set the $Z(t)$- and the $B_c(t)$-variables. These variables are constrained by the $\delta_c(t)$-constraint in the dual. By Equation (3.7) it is sufficient to fulfill, for any $\tau \in I$

$$(\alpha^+(I) - \alpha^-(I))f_{\max} + \mathrm{vol}_c^N(\tau) \le Z(\tau) + B_c(\tau) - B_c(\tau-1) \ .$$

We do this while ensuring $B_c(\tau) \le V$ for all τ, i.e., the $x_c(\tau)$-constraints are violated by at most a factor of V. We set $Z(\tau_i) = V + (\alpha^+(\tau_i) - \alpha^-(\tau_i))f_{\max}$ and $B_c(\tau_i) = \mathrm{vol}_c^N(\tau_i)$. Then the above constraint is fulfilled for $\tau = \tau_i$ as $B_c(\tau_i - 1) \le V$ by induction. For the other time steps in I we fulfill the constraint by setting $Z(\tau) = (\alpha^+(\tau) - \alpha^-(\tau))f_{\max}$ and by setting $B_c(\tau) = B_c(\tau-1) + \mathrm{vol}_c^N(\tau)$. The B_c-variables are therefore increasing over interval I with $B_c(\tau_{i+2}-1) = \mathrm{vol}_c^N(I) \le V$. This fulfills the $\delta_c(t)$-constraints.

We get that $\sum_{t\in I} Z(t) = V + (\alpha^+(I) - \alpha^-(I))f_{\max}$, so the cost control budget for interval I is

$$\mathcal{B}_{\mathrm{control}}(I) = \sum_{t\in I}\sum_c \Big(\mathrm{vol}_c^N(t) - \Delta_c(t)\Big) - kV \ . \tag{3.8}$$

We now make a case distinction on the colors. There are $k + n_{i+1}$ colors in $M_i \cup M_{i+1}$, each having $\mathrm{vol}_c^N(I) = V$. Exactly $2n_{i+1}$ of these colors are unmarked in one of the phases.

Recall that the *gap* is defined as $\Delta_c(I) := \mathrm{vol}_c^T(I) - \mathrm{vol}_c^E(I) \leq \gamma \mathrm{vol}_c^E(I)$. An unmarked color collects volume at most $6V$, hence its gap is at most $6\gamma V$ by Equation (3.5). The gap generated by all marked colors throughout both phases is at most $10\gamma(n_i + n_{i+1})$ by Condition III.

A color that is unmarked in both phases (i.e., a color not in $M_i \cup M_{i+1}$) has $\mathrm{vol}_c^T(I) \leq 12V$ by Condition II, and, hence, $\mathrm{vol}_c^N(I) \geq \frac{1}{12}\mathrm{vol}_c^T(I)$ for these colors. Its gap is at most $\gamma \mathrm{vol}_c^T(I)$, a γ-fraction of its total volume.

Plugging these observations into Equation (3.8) give a cost control budget of at least

$$\mathcal{B}_{\text{control}}(I) \geq n_{i+1}V + \sum_{c \in R} \big(\tfrac{1}{12} - \gamma\big)\mathrm{vol}_c^T(I) - 10\gamma n_i V - 22\gamma n_{i+1} V \ , \tag{3.9}$$

where R denotes the set of colors not in $M_i \cup M_{i+1}$. Let U_i denote the set of color that are unmarked in the i-th phase. Each of these colors has $\mathrm{vol}_c^T(I_i) \leq 6V$, hence

$$\sum_{c \in U_i \setminus R} \big(\tfrac{1}{12} - \gamma\big)\mathrm{vol}_c^T(I_i) \leq \tfrac{1}{2} n_{i+1} V \ ,$$

because $|U_i \setminus R| = n_{i+1}$ as each of these colors is new in phase $i+1$. Plugging this into Equation (3.9) gives

$$\begin{aligned} \mathcal{B}_{\text{control}}(I) &\geq \tfrac{1}{2} n_{i+1} V + \sum_{c \in U_i} \big(\tfrac{1}{12} - \gamma\big)\mathrm{vol}_c^T(I_i) - 10\gamma n_i V - 22\gamma n_{i+1} V \\ &\geq \tfrac{1}{4} n_{i+1} V + \sum_{c \in U_i} \tfrac{1}{24}\mathrm{vol}_c^T(I_i) - \tfrac{1}{12} n_i V \ , \end{aligned} \tag{3.10}$$

where we used $\gamma = 1/128$ and

$$\begin{aligned} \sum_{c \in R} \mathrm{vol}_c^T(I) &= \sum_{c \in U_i} \mathrm{vol}_c^T(I_i) - \sum_{c \in U_i \setminus R} \mathrm{vol}_c^T(I_i) + \sum_{c \in U_i} \mathrm{vol}_c^T(I_{i+1}) - \sum_{c \in U_i \setminus R} \mathrm{vol}_c^T(I_{i+1}) \\ &\geq \sum_{c \in U_i} \mathrm{vol}_c^T(I_i) - \sum_{c \in U_i \setminus R} \mathrm{vol}_c^T(I_i) \ . \end{aligned}$$

In order to obtain a lower bound over the full time horizon, we group phases $(1,2)$, $(3,4)$,$(5,6),\ldots$ and $(2,3),(4,5),(6,7),\ldots$ and taking the average over the two sequences. This gives a lower bound for the overall cost control budget of

$$\mathcal{B}_{\text{control}} \geq \sum_i \tfrac{1}{16} n_i V + \sum_i \sum_{c \in U_i} \tfrac{1}{48}\mathrm{vol}_c^T(I_i) - \tfrac{1}{8} n_1 V - \sum_{c \in U_\ell} \tfrac{1}{48}\mathrm{vol}_c^T(I_\ell) \ .$$

Note that Phase 1 does not appear as a second phase in a group and Phase ℓ (the last phase) does not appear as first phase in a group. Therefore, the corresponding contributions of these phases are subtracted in the above sum.

The sum of cost control budget and base budget $\mathcal{B}_{\text{control}} + \mathcal{B}_{\text{base}}$ give the total profit of the dual program. In order to obtain a lower bound, we divide it by the violation W of the $y_c(\tau)$-constraints. The terms depending on n_1 and U_ℓ are removed by exploiting the additional lower bound $\text{OPT} \geq \frac{1}{2}(n_1 + |U_\ell|)$, which holds as every color needs to be accessed at least once. As $V < W$ and the total volume of an unmarked color is at most $6V$, the terms disappear by averaging. By choosing a sufficiently small constant $\xi > 0$, we obtain the lemma. □

We show later that we can keep the violation W in $\Theta(V)$. Further more, the deactivation procedure only deactivates colors that have collected total volume at least V in their current activity period. This gives us the following simpler lower bounds.

Corollary 3.8. *If $W \in \Theta(V)$, then the following three lower bounds on the optimal solution* OPT *hold:*

$$\text{OPT} \geq \Omega\left(\frac{1}{2W}\int_0^{\alpha_{\max}} \Big(|E(\vec{z},t)| - b'\Big)\,\mathrm{d}\alpha\right) ,$$
$$\text{OPT} \geq \Omega\big(\textstyle\sum_i n_i\big) ,$$
$$\text{OPT} \geq \Omega\big(\textstyle\sum_c a_c\big) .$$

Here a_c denotes the number of activity periods of color c for which c is a non-stale color at the time of activation.

Proof. The first two lower bounds are immediate from Lemma 3.7. We prove $\text{OPT} \geq \Omega\big(\sum_c a_c\big)$.

Fix a color c and an activity period P during which c is non-stale at the time of activation. Suppose there is an activity period Q of c following P. If the color is marked during P or Q, it is a new color during that phase and therefore contributes $\Omega(1)$ to the lower bound. Otherwise, suppose P spans phases i up to j. Color c is not marked during phase i, as it would be a stale color at the start of P otherwise. The color is also not marked during phase $i+1 \ldots, j$, as its only activity periods during these phases are P and Q. Hence, color c is deactivated at the end of P after having collected volume at least V (Algorithm 3). This volume fully contributes to the lower bound, so the contribution of P is $\Omega(V/W) = \Omega(1)$. Any two consecutive activity periods therefore give a constant contribution to the lower bound, which gives the claim. □

3.3.2. Analysis of the base procedure

We now give the analysis of the base procedure and we specify the randomized rounding algorithm. The proofs in this section are nearly identical to the work of Adamaszek et al. [ACER12] and they are included for completeness. Aside from minor notational differences, the only change is the inclusion of $\delta_c(\tau)$-variables in many statements. This modification does not structurally impact the proofs.

Analyzing the cost of the base procedure

We start by showing that the dual constraints $y_c(\tau)$ of LP Dual are not violated through the increase of variables $\alpha_1(\vec{v}, t)$.

Lemma 3.9. *For every c and τ, $W_{\text{base}} := \sum_{\vec{v},t' \geq \tau} |E_c(\vec{v},\tau)|_{t'} \cdot \alpha_1(\vec{v},t') = \mathcal{O}(\log\log b)$.*

Proof. Let $\chi_c(\tau) = \sum_{\vec{v},t' \geq \tau} |E_c(\vec{v},\tau)|_{t'} \cdot \alpha_1(\vec{v},t') - 1$ denote the amount by which the dual constraint $y_c(\tau)$ is violated. $\chi_c(\tau)$ can increase if some dual variable $\alpha_1(\vec{z},t)$ is increased. However, note that this only causes an increase of $\chi_c(\tau)$ if $z_c < \tau \leq t$. Further note that our procedure only increases $\alpha_1(\vec{z},t)$ if the corresponding primal constraint is proper, which implies $\sum_{z_c < \tau \leq t} y_c(\tau) < 2/\beta$. Therefore, once $\sum_{\tau < i \leq t} y_c(i) \geq 2/\beta$, $\chi_c(\tau)$ does not increase any further. In the following we analyze how large $\chi_c(\tau)$ can become before $\sum_{\tau < i \leq t} y_c(i) \geq 2/\beta$. We have the following claim.

Claim 3.10. *A violated dual constraint $y_c(\tau)$ fulfills $\sum_{\tau < i \leq t} y_c(i) \geq e^{\chi_c(\tau)}/\log^3 b$.*

Proof. When the dual constraint $y_c(\tau)$ becomes tight (i.e., $\chi_c(\tau) = 0$) in some time step t, our procedure sets $y_c(t)$ to at least $1/\log^3 b$. Therefore, at this point

$$\sum_{\tau < i \leq t} y_c(i) \geq e^{\chi_c(\tau)}/\log^3 b \tag{3.11}$$

as required.

In later time time steps, an increase of dual variable $\alpha_1(\vec{z},t')$ by $\mathrm{d}\alpha$ increases the violation $\chi_c(\tau)$ by $|E_c(\vec{z},\tau)|_{t'} \cdot \mathrm{d}\alpha$. This means that the right hand side of Equation (3.11) increases by

$$\frac{e^{\chi_c(\tau)}}{\log^3 b} \cdot |E_c(\vec{z},\tau)|_{t'} \cdot \mathrm{d}\alpha \ .$$

However, at the same time $\sum_{\tau < i \leq t'} y_c(i)$ increases by at least

$$\sum_{\tau < i \leq t} y_c(i) \cdot |E_c(\vec{z},\tau)|_t \cdot \mathrm{d}\alpha \geq \frac{e^{\chi_c(\tau)}}{\log^3 b} \cdot |E_c(\vec{z},\tau)|_{t'} \cdot \mathrm{d}\alpha$$

□

The previous claim implies that the violation of a dual constraint can grow to at most $\mathcal{O}(\log\log b)$ before $\sum_{\tau<i\leq t'} y_c(i)$ becomes $2/\beta$ after which the violation does not increase any further. □

Our next goal is to derive a bound on the total amount by which y-variables are increased in the base procedure. Recall that $\hat{y}$ is defined in the base procedure (Algorithm 1) and is either 0 or $1/\log^3 b$. The following two technical lemmas will be useful.

Lemma 3.11. *If $\alpha_1(\vec{v},t) > 0$, then $\sum_c \sum_{\tau\leq t} |E_c(\vec{v},\tau)|_t \cdot \hat{y}_c(\tau) \leq |E(\vec{v},t)| - b'$.*

Proof. Fix a color c. We partition the time steps $\tau \in \{v_c+1,\ldots,t\}$ into classes according to the value of $|E_c(\vec{v},\tau)|$. For $j \geq 0$, the class $T_{c,j}$ contains the time steps for which $|E_c(\vec{v},\tau)| \in [2^j, 2^{j+1})$. Let J_c be the maximum value of j so that class $T_{c,j}$ is nonempty.

Claim 3.12. *A class $T_{c,j}$ includes at most $\mathcal{O}(\log\log b)$ time steps τ for which $\hat{y}_c(\tau) > 0$.*

Proof. Since $\alpha_1(\vec{v},t) > 0$ the primal constraint $\alpha(\vec{v},t)$ was proper and violated at time t.

Fix a class $T_{c,j}$ and let $\tau_1, \tau_2, \ldots$ denote its time steps that have $\hat{y}_c(\tau_i) > 0$ in increasing order. Define $L(\tau) := \sum_{\vec{v},t'\geq\tau} |E_c(\vec{v},\tau)|_{t'} \cdot \alpha_1(\vec{v},t')$ to be the load of τ (the left hand side of the constraint $y_c(\tau)$ in the dual). Note that for $\tau \in T_{c,j}$ with $\tau \geq \tau_1$, an increase of $L(\tau)$ by $|E_c(\vec{v},\tau)|_t \cdot \alpha_1(\vec{v},t)$ also results in an increase of $L(\tau_1)$ by $|E_c(\vec{v},\tau_1)|_t \cdot \alpha_1(\vec{v},t)$ which is the same up to a factor of 2 as τ_1 and τ_i are in the same class. In order for $\hat{y}_c(\tau_i)$ to be set to a positive value, some $L(\tau)$ for $\tau_{i-1} < \tau \leq \tau_i$ has to have increased to 1 (as a constraint needs to be tight). Hence, for every τ_i with $\hat{y}_c(\tau_i) > 0$, the load $L(\tau_1)$ will increase by at least $1/2$. But due to Lemma 3.9, the load $L(\tau_1)$ is at most $\mathcal{O}(\log\log b)$. □

Let $n_{c,j}$ denote the number of time steps τ in class $T_{c,j}$ that have $\hat{y}_c(\tau) > 0$. Then we have

$$\begin{aligned}\sum_c \sum_{\tau\leq t} |E_c(\vec{v},\tau)|_t \cdot \hat{y}_c(\tau) &\leq \sum_c \sum_{j>0}^{J_c} 2^{j+1} \frac{n_{c,j}}{\log^3 b} \\ &\leq \sum_c \mathcal{O}(\log\log b) \cdot \frac{|E_c(\vec{v},t)|}{\log^3 b} \\ &\leq \frac{|E(\vec{v},t)|}{\log b} \leq |E(\vec{v},t)| - b' \ .\end{aligned}$$

The second inequality holds as $n_{c,j} \leq \mathcal{O}(\log\log b)$ by claim and $\sum_{j>0}^{J_c} 2^{j+1} \leq 4 \cdot 2^{J_c} \leq 8|E_c(\vec{v},t)|$. □

Lemma 3.13. *If* $\hat{y}_c(\tau) > 0$*, then* $\sum_{\vec{v},t\geq\tau} |E_c(\vec{v},\tau)|_t \cdot \alpha_1(\vec{v},t) \geq 1$.

Proof. $\hat{y}_c(\tau)$ is only set to a non-zero value if the dual constraint $y_c(\tau)$ is tight. Because α_1-variables are never decreased, once the constraint is tight it can only remain tight or become violated. □

We are now ready to derive our desired bound on $\sum_{c,t} \mathrm{d}y_c(t)$.

Lemma 3.14. *The cost of the base procedure is at most* $2\sum_{\vec{v},t} \alpha_1(\vec{v},t)\big(|E(\vec{v},t)| - b'\big)$.

Proof. The cost of the base procedure $\sum_{c,t} \mathrm{d}y_c(t)$ consists of two components: $\sum_{c,t} \hat{y}_c(t)$ and $\sum_{c,t} \max_{\tau:z_c<\tau\leq t} \Delta(\tau,c)$. We bound both components separately.

Using Lemma 3.11 and then Lemma 3.13, we get

$$\begin{aligned}\sum_{\tau,c} \hat{y}_c(\tau) &\leq \sum_{\tau,c} \hat{y}_c(\tau) \cdot \sum_{\vec{v},t\geq\tau} |E_c(\vec{v},\tau)|_t \cdot \alpha_1(\vec{v},t) \\ &= \sum_{\vec{v},t} \alpha_1(\vec{v},t) \sum_c \sum_{\tau\leq t} |E_c(\vec{v},\tau)|_t \hat{y}_c(\tau) \\ &\leq \sum_{\vec{v},t} \alpha_1(\vec{v},t)\big(|E(\vec{v},t)| - b'\big) \ .\end{aligned}$$

For the second component, we have for a time step t,

$$\begin{aligned}\sum_{c,t} \max_{\tau:z_c<\tau\leq t} \Delta(\tau,c) &= \sum_c \max_{\tau:z_c<\tau\leq t} \sum_{\tau\leq i\leq t} y_c(i) \cdot |E_c(\vec{z},\tau)|_t \,\mathrm{d}\alpha \\ &\leq \sum_c \sum_{\tau\leq t} y_c(\tau) \cdot |E_c(\vec{z},\tau)|_t \,\mathrm{d}\alpha \\ &\leq \sum_c \sum_{\tau\leq t} y_c(\tau) \cdot |E_c(\vec{z},\tau)|_t \,\mathrm{d}\alpha_1(\vec{z},t) \ ,\end{aligned}$$

where the last step follows because $\alpha_1(\vec{z},t)$ is increased by $\mathrm{d}\alpha$. Because the primal $\alpha(\vec{z},t)$ constraint is violated, this is at most $(|E(\vec{z},t)| - b')\,\mathrm{d}\alpha_1(\vec{z},t)$. Summing over all time steps t gives the desired bound. □

Randomized rounding

The goal of this section is to show to provide a randomized rounding algorithm for the base procedure.

The y-variables, some of which are set by the base procedure and some of which are set equal to 1 by the cost control procedure can, in fact, be rounded online in such a way that the buffer constraint is satisfied. This is true independent of which (if any)

y-variables are set to 1 by the cost control procedure and also independent of the values of the δ-variables.

A deterministic strategy for scheduling block operations is given by a function $Q : T \times C \to \mathbb{N}_0$ with the meaning that $Q(\tau, c)$ describes the number of block operations for color c performed in time step τ. We allow $Q(\tau, c) > 1$, i.e., we will allow that the strategy performs several consecutive block operations for the same color.

We specify a number of conditions on a deterministic strategy. The conditions are chosen in such a way that any deterministic strategy that satisfies them produces an overall solution that ensures that the buffer never stores more than b items.

In order to formulate the conditions for time t, we introduce the following notation. We use $w_c(\tau) := \max\{\beta y_c(\tau), 1\}$ to denote a scaled version of the current assignment to y-variables in the primal LP. We partition the pairs (τ, c) with $\tau \in \{z_c + 1, \ldots, t\}$ into classes according to the value of $|E_c(\vec{z}, \tau)|$. Here and in the following $\vec{z}$ is to be understood as the vector that our procedure maintains. We say a pair (τ, c) is in class S_i if $|E_c(\vec{z}, \tau)| \in [2^i, 2^{i+1})$ (we do not care about (τ, c)-pairs that have $|E_c(\vec{z}, \tau)| = 0$). We further use S^c to denote the set $\{(\tau, c) \mid \tau \in \{z_c + 1, \ldots, t\}\}$ and $S = \bigcup_c S^c$.

In addition to sets S_i we also define a set L that contains (τ, c)-pairs with a "large" $|E_c(\vec{z}, \tau)|$-value. Formally, we first select pairs in decreasing order of $|E_c(\vec{z}, \tau)|$-value until L contains pairs whose $w_c(\tau)$-values sum up to at least $\lambda + 1$ (for a parameter $\lambda \ll \beta$ to be chosen later) or $L = S$; then if the $w_c(\tau)$-values sum up to more than $\lambda + 1$ we remove the last element added. Hence, if $L \neq S$ we have $\lambda \leq \sum_{(\tau,c)\in L} w_c(\tau) \leq \lambda + 1$, as the $w_c(\tau)$'s are at most 1.

Our conditions for a deterministic strategy Q are as follows.

(A) For every color c, $\sum_{z_c \leq \tau \leq t}(Q(\tau, c) + \delta_c(\tau)) \geq 1$, i.e., between time z_c and t, all items of color c are removed from the buffer at least once; either because Q performed at least one block device operation or because at some point one of the k servers was located at c.

(B) Q mirrors the fractional solution of LP Primal on the sets S_i:

$$\forall S_i : \quad \lfloor \sum_{(\tau,c)\in S_i} w_c(\tau) \rfloor \leq \sum_{(\tau,c)\in S_i} Q(\tau, c) \ .$$

(C) Q mirrors the fractional solution of LP Primal on the set L of *large* (τ, c)-pairs:

$$\lfloor \sum_{(\tau,c)\in L} w_c(\tau) \rfloor \leq \sum_{(\tau,c)\in L} Q(\tau, c) \ .$$

(D) For every color c and for every class S_i, $\sum_{(\tau,c)\in S_i\cap S^c} Q(\tau,c) \leq 3$. This means Q did not remove the same color very often in the same class.

Suppose these conditions hold at the end of time t, where $\vec{z}$ is the specific vector maintained by our base procedure, i.e., $\alpha(\vec{z},t)$ is a proper constraint. Further suppose that the proper constraint $\alpha(\vec{z},t)$ is satisfied. In this case we will show next that the buffer contains no more than b items.

Condition (A) guarantees that items of color c that appeared at time z_c or before do not influence the buffer-constraint for Q at time t, since all these items have already been removed from the buffer. Hence, the following formula specifies exactly the number of items in Q's buffer at time t:

$$\text{buffer}(t) = |E(\vec{z},t)| - \sum_c \max_{\tau:Q(\tau,c)+\delta_c(\tau)\geq 1} |E_c(\vec{z},\tau)| \ .$$

This holds because $|E_c(\vec{z},\tau)|$ (for the maximum τ with $Q(\tau,c)+\delta_c(\tau)\geq 1$) specifies the items that appeared after time z_c (excluding z_c) and are evicted at time τ or before. Furthermore, remember that if $\alpha(\vec{z},t)$ is a proper constraint, we must have $\delta_c(\tau)=0$ for all c and $\tau > z_c$, i.e., there is no server at color c after time z_c. We therefore also have

$$\text{buffer}(t) = |E(\vec{z},t)| - \sum_c \max_{\tau:Q(\tau,c)\geq 1} |E_c(\vec{z},\tau)| \ . \tag{3.12}$$

Let j denote the index of the largest class that contains a pair (τ,c) with $Q(\tau,c)+\delta_c(\tau)\geq 1$. Then

$$\begin{aligned}\max_{\tau:Q(\tau,c)\geq 1} |E_c(\vec{z},\tau)| \geq 2^j &\geq \frac{1}{12}\sum_{i=0}^{j} 3\cdot 2^{i+1} \\ &\geq \frac{1}{12}\sum_{(\tau,c)\in S^c} Q(\tau,c)\cdot |E_c(\vec{z},\tau)| \ ,\end{aligned}$$

where the last step uses Condition (D) (the fact that Q does not evict a color too often in the same class). Plugging the above into Equation (3.12) gives

$$\text{buffer}(t) \leq |E(\vec{z},t)| - \frac{1}{12}\sum_{(\tau,c)\in S} Q(\tau,c)\cdot |E_c(\vec{z},\tau)| \ .$$

We need to show that this is at most b. This means we want to show that

$$\sum_{(\tau,c)\in S} Q(\tau,c)\cdot|E_c(\vec{z},\tau)| \geq 12(|E(\vec{z},t)|-b) \ . \tag{3.13}$$

This is encapsulated in the following lemma.

Lemma 3.15. *Let $\alpha(\vec{z},t)$ be a proper, non-violated constraint. A scheduling strategy that fulfills conditions A, B, C, and D fulfills $\sum_{(\tau,c)\in S} Q(\tau,c)\cdot|E_c(\vec{z},\tau)| \geq 12(|E(\vec{z},t)|-b)$.*

Proof. Any $\alpha(\vec{z},t)$-constraint of LP Primal fulfills $|E_c(\vec{z},t)| \geq b$. We first show that this implies the following technical claim.

Claim 3.16. $\lambda \leq \sum_{(\tau,c)\in L} w_c(\tau) \leq (\lambda+1)$.

Proof. If is sufficient to show that $\lambda \leq \sum_{z_c<\tau\leq t} w_c(\tau)$, then by definition of the set L, the sum will be at most $\lambda+1$.

The primal constraint $\alpha(\vec{z},t)$ is fulfilled. Therefore,

$$\sum_c \Big(\sum_{z_c<\tau\leq t} |E_c(\vec{z},\tau)|_t \cdot y_c(\tau) + |E_c(\vec{z},t)|_t \cdot \delta_c(t) \Big) \geq |E(\vec{z},t)| - b' \ .$$

The left term is equal to $\sum_c \sum_{z_c<\tau\leq t} |E_c(\vec{z},\tau)|_t \cdot y_c(\tau)$ because whenever $\delta_c(t)=1$ we have $z_c = t$ and hence $|E_c(\vec{z},t)|_t = 0$. Since $|E(\vec{z},t)| \geq b$, we also have $|E_c(\vec{z},\tau)|_t \leq |E(\vec{z},t)|-b'$ and therefore, the $y_c(\tau)$ must sum to at least one. As $w_c(\tau) \geq \beta y_c(\tau)$, this gives that the $w_c(\tau)$'s sum up to at least β. Hence, L contains a $w_c(\tau)$-weight of at least λ as $\lambda \ll \beta$. □

The above claim then gives $\sum_{(\tau,c)\in L} w_c(\tau) \geq \lambda$, hence with Condition (C)

$$\sum_{(\tau,c)\in L} D(\tau,c) \geq \lambda \ .$$

Now assume that i_ℓ, the class-index of the pair $(\tau,c) \in L$ with smallest $|E_c(\vec{z},\tau)|$-value, fulfills $2^{i_\ell} \geq |E(\vec{z},t)| - b$. Then

$$\begin{aligned}
\sum_{(\tau,c)\in S} |E_c(\vec{z},\tau)|\cdot Q(\tau,c) &\geq \sum_{(\tau,c)\in L} |E_c(\vec{z},\tau)|\cdot Q(\tau,c) \\
&\geq 2^{i_\ell} \sum_{(\tau,c)\in L} Q(\tau,c) \\
&\geq \lambda(|E(\vec{z},t)|-b) \\
&\geq 12(|E(\vec{z},t)|-b) \ ,
\end{aligned}$$

by choosing $\lambda = 12$. This gives the lemma if $2^{i_\ell} \geq |E(\vec{z},t)| - b$.

We now assume $2^{i_\ell} \leq (|E(\vec{z},t)| - b)$. We have

$$\begin{aligned}\sum_{(\tau,c)\in S} |E_c(\vec{z},\tau)|Q(\tau,c) &\geq \sum_{i\leq i_\ell} \sum_{(\tau,c)\in S_i} |E_c(\vec{z},\tau)|Q(\tau,c) \\ &\geq \sum_{i\leq i_\ell} 2^i \sum_{(\tau,c)\in S_i} Q(\tau,c) \\ &\geq \sum_{i\leq i_\ell} 2^i \Big(\sum_{(\tau,c)\in S_i} w_c(\tau) - 1 \Big) \\ &= \frac{1}{2} \sum_{i\leq i_\ell} \sum_{(\tau,c)\in S_i} 2^{i+1} w_c(\tau) - \sum_{i\leq i_\ell} 2^i \ .\end{aligned}$$

It follows that

$$\begin{aligned}\sum_{(\tau,c)\in S} |E_c(\vec{z},\tau)|Q(\tau,c) &\geq \frac{1}{2} \sum_{i\leq i_\ell} \sum_{(\tau,c)\in S_i} |E_c(\vec{z},\tau)|w_c(\tau) - 2^{i_\ell+1} \\ &\geq \frac{\beta}{2} \sum_{i\leq i_\ell} \sum_{(\tau,c)\in S_i} |E_c(\vec{z},\tau)|_t y_c(\tau) - 2(|E(\vec{z},t)| - b) \ .\end{aligned} \tag{3.14}$$

Then

$$\begin{aligned}\frac{\beta}{2} \sum_{L\setminus S_{i_\ell}} |E_c(\vec{z},\tau)|_t y_c(\tau) &\leq \frac{1}{2} \sum_{(\tau,c)\in L} |E_c(\vec{z},\tau)|_t w_c(\tau) \\ &\leq (|E(\vec{z},t)| - b) \sum_{(\tau,c)\in L} w_c(\tau) \\ &\leq (\lambda+1)(|E(\vec{z},t)| - b)\end{aligned}$$

where the last step uses $\sum_{(\tau,c)\in L} w_c(\tau) \leq \lambda + 1$ by Claim 3.16. Adding the inequality $0 \geq \frac{\beta}{2} \sum_{L\setminus S_{i_\ell}} |E_c(\vec{z},\tau)|_t y_c(\tau) - (\lambda+1)(|E(\vec{z},t)| - b)$ to Equation (3.14) we obtain

$$\begin{aligned}\sum_{(\tau,c)\in S} |E_c(\vec{z},\tau)|Q(\tau,c) &\geq \frac{\beta}{2} \sum_{(\tau,c)\in S} |E_c(\vec{z},\tau)|_t y_c(\tau) - (\lambda+3)(|E(\vec{z},t)| - b) \\ &\geq 12(|E(\vec{z},t)| - b) \ ,\end{aligned}$$

where the last inequality holds as $\beta = 80$, $\lambda = 12$, and the fact that the primal constraint for $\alpha(\vec{z},t)$ is fulfilled and either $\delta_c(t) = 0$ or $|E_c(\vec{z},t)|_t = 0$. □

It remains to show how to update the distribution in an online manner so that the block operation strategies fulfill conditions A, B, C, and D.

Instead of constructing μ directly we will first construct distributions μ_1, μ_2, and μ_3. A random buffer management strategy according to μ is then chosen by sampling strategies $\mu_1 \sim Q_1$, $\mu_2 \sim Q_2$, $\mu_3 \sim Q_3$ and computing the strategy Q by setting

$$Q(\tau, c) := \max\{Q_1(\tau, c), Q_2(\tau, c), Q_3(\tau, c)\} \ .$$

Any strategy, whether in the support of μ_1, μ_2, or μ_3 fulfills for all c and S_i that $\sum_{(\tau,c)\in S_i \cap S^c} Q(\tau, c) \leq 3$. Then it is clear that the strategy $Q = \max\{Q_1, Q_2, Q_3\}$ fulfills $\forall c, \forall S_i \sum_{(\tau,c)\in S_i \cap S^c} Q(\tau, c) \leq 3$ (Condition (D)).

In addition, strategies in the support of μ_1 fulfill Condition (A), strategies from μ_2 fulfill Condition (B), and strategies from μ_3 fulfill Condition (C). Hence, Q will fulfill all these conditions and consequently Q will obey buffer-constraints.

Maintaining μ_1, μ_2, and μ_3. In the following we describe how to update μ_1, μ_2, and μ_3, and make sure that the strategies in their support fulfill their respective conditions. The distributions μ_1 and μ_2 will always fulfill $\sum_Q \mu_i(Q) \cdot Q(\tau, c) = w_c(\tau)$, i.e., the expected number of times the color c is removed at time τ by a random strategy Q is equal to the scaled LP-variable $w_c(\tau)$ (recall that we allow a strategy to remove a color several times in the same time step). It would be difficult to maintain the above property without allowing changes in the past. Therefore, we will allow a strategy at time t to alter its past behavior and to increase or decrease $Q(\tau, c)$ for $\tau < t$ (such a change may incur a cost, of course). In reality, decreasing a $Q(\tau, c)$-value only makes fulfilling buffer-constraints more difficult, so allowing this does not give additional power to the algorithm. Similarly, increasing a $Q(\tau, c)$-value with $\tau < t$ is never better than increasing $Q(t, c)$ instead, since we only care about the buffer-constraint at t as the others have already been met.

There are two types of changes that require updating the distributions. Firstly, the $w_c(\tau)$-values may increase. We will assume that these changes are infinitesimal, i.e., in each time step we have to react to an increase of a $w_c(\tau)$-value from $w_c(\tau)$ to $w_c(\tau) + \varepsilon$.

The other type of change is a change to the vector $\vec{z}$. When $\vec{z}$ is recomputed because otherwise the primal $\alpha(\vec{z}, t)$ constraint is not proper anymore, we set, for each color c for which one of the conditions is violated, z_c to the largest possible value (less or equal to t) such that $\sum_{z_c \leq \tau \leq t}(y_c(\tau) + \delta_c(\tau)) \geq 1/\beta$. The above procedure guarantees that the following properties *always* hold

- $\sum_{(\tau,c):z_c<\tau\leq t} w_c(\tau) < 3$. A primal $\alpha(\vec{z}, t)$ constraint is proper if $\sum_{(\tau,c):z_c<\tau\leq t} w_\tau(c) \leq 2$. The base procedure (Algorithm 1), increases w-values by at most $\beta/\log^3 b$ within a single time step. For large enough b this means that the above sum never exceeds 3.

- $\sum_{z_c \le \tau \le t} w_c(\tau) \ge 1$. In the end we want that every strategy Q from μ has every color removed at least once within the range $\{z_c, \ldots, t\}$. Therefore this property is important.

Whenever $\vec{z}$ changes $\sum_{(\tau,c):z_c<\tau\le t} w_c(\tau)$ decreases by at least 1. Therefore a fixed pair (τ, c) is only involved in at most 3 different $\alpha(\vec{z}, t)$ constraints.

For increasing $w_c(\tau)$-values we show that an increase in ε results in an expected cost of $\mathcal{O}(\varepsilon)$ for the maintenance operation. We also implement a maintenance operation with $\mathcal{O}(\varepsilon)$ expected cost when a $w_c(\tau)$-value decreases. With this decrement operation we implement a change of the vector $\vec{z}$ as follows.

Suppose we want to change the c-th coordinate of $\vec{z}$ from z_c to z_c'. This may make all conditions that depend on the values $|E_c(\vec{z}, \tau)|$ invalid. We decrement all $w_c(\tau)$-values with $\tau \in \{z_c' + 1, \ldots, t\}$ to 0. Then we change z_c to z_c', and after that we increase the $w_c(\tau)$-values again. The cost for this is $\mathcal{O}(w_c(\tau))$ for every (τ, c)-pair involved. Since each (τ, c)-pair is only involved in a constant number of these decrement operations we obtain that the total cost for the change is only $\mathcal{O}(\sum_{\tau,c} w_c(\tau)) = \mathcal{O}(\sum_{t,c} y_c(t))$, provided that we can implement the increase and decrease operations as claimed.

Maintaining μ_1. Recall Condition (A) states, for every c, $\sum_{z_c \le \tau \le t}(Q(\tau, c) + \delta_c(\tau)) \ge 1$.

For maintaining μ_1 we do not require a decrement operation as Condition (A) does not depend on $|E_c(\vec{z}, \tau)|$-values. Suppose that $w_c(\tau)$ increases by ε. Then we first identify an ε-measure of strategies that have the smallest value of $\arg\max_{\tau'}\{Q(\tau', c) > 0\}$, i.e., strategies that did not remove color c for a long time. For these strategies we set $Q(\tau, c) = 1$. This means that the strategies evict color c in a round-robin fashion. Since $\sum_{(\tau,c)\in S^c} \sum_Q \mu(Q) Q(\tau, c) = \sum_{(\tau,c)\in S^c} w_\tau(c) \ge 1$, Condition (A) follows.

Maintaining μ_2. We maintain a *strengthening* of Condition (B), namely that for all S_i

$$\left\lfloor \sum\nolimits_{(\tau,c)\in S_i} w_c(\tau) \right\rfloor \le \sum\nolimits_{(\tau,c)\in S_i} Q(\tau, c) \le \left\lceil \sum\nolimits_{(\tau,c)\in S_i} w_c(\tau) \right\rceil . \tag{3.15}$$

Suppose that the $w_{\bar{c}}(\bar{\tau})$ value for some pair $(\bar{\tau}, \bar{c})$ is increased by ε and assume that $(\bar{\tau}, \bar{c}) \in S_i$. As we want to satisfy $\sum_Q \mu_1(Q) Q(\bar{\tau}, \bar{c}) = w_{\bar{c}}(\bar{\tau})$ we have to increase $Q(\bar{\tau}, \bar{c})$ for some strategies. For this we choose an ε-fraction of strategies that have $\sum_{(\tau,c)\in S^c \cap S_i} Q(\tau, c) \le 2$. Observe that such strategies must exist for small enough ε as

$$\sum_Q \sum_{(\tau,c)\in S^c\cap S_i} Q(\tau, c)\mu_1(Q) = \sum_{(\tau,c)\in S^c\cap S_i} w_c(\tau) \le \sum_{(\tau,c)\in S^c} w_c(\tau) < 3$$

. We increase the value of $Q(\tau, c)$ for the chosen strategies.

Suppose the constraint in Equation (3.15) is be violated for some class S_i. Let $a = \lfloor \sum_{(\tau,c)\in S_i} w_c(\tau) \rfloor$ (before changing $w_c(\tau)$) and first assume that $\lfloor \sum_{(\tau,c)\in S_i} w_c(\tau) \rfloor$ remains equal to a. Then the strategies that just have been changed may now have $\sum_{(\tau,c)\in S_i} Q(\tau, c) = a + 2$, which is not allowed.

We match these strategies to strategies that have $\sum_{(\tau,c)\in S_i} Q(\tau, c) = a$. For each strategy Q there must exist a (τ, c) such that $Q(\tau, c) > Q'(\tau, c)$, where Q' denotes the strategy that Q is matched to. We decrease $Q(\tau, c)$ by 1 and increase $Q'(\tau, c)$ by 1. This only induces an expected cost of $\mathcal{O}(\varepsilon)$. The case in which $\lfloor \sum_{(\tau,c)\in S_i} w_c(\tau) \rfloor$ changes is analogous.

Also the decrement operation can be implemented this way. When $w_c(\tau)$ decreases we select an ε-measure of strategies that fulfill $Q(\tau, c) > 0$ and decrease $Q(\tau, c)$ for them. Then we do a re-balancing step.

Maintaining μ_3. Here we maintain a strengthened version of Condition (C). Let $(\tau_1, c_1), (\tau_2, c_2), \ldots$ denote the sequence of (τ, c)-pairs from S, in decreasing order of $|E_c(\vec{z}, \tau)|$. Let (τ_r, c_r) denote the first pair in this sequence that is *not* in L (note that this may not exist; then we define r as $|S| + 1$ since $L = S$). Define a function ℓ on the (τ, c)-pairs that is zero for all (τ_i, c_i), $i > r$; one for all (τ_i, c_i), $i < r$ and $w_{c_r} - (\sum_{i=1}^{r} w_{c_i}(\tau_i) - (\lambda + 1))/w_{c_r}(\tau_r)$ for (τ_r, c_r). For all (τ_i, c_i), $i \neq r$ ℓ simply is the characteristic function of the set L; only for r it measures the fraction by which (τ_r, c_r) needs to be included into L in order that the $w_c(\tau)$-values in L sum up to *exactly* $(1 + \lambda)$ (recall that during the construction of L we were aiming for it to contain a total w-weight of $\lambda + 1$. When we overshoot we remove the last element).

We maintain the constraint that

$$\Big\lfloor \sum\nolimits_{(\tau,c)\in S} w_c(\tau)\ell(\tau, c) \Big\rfloor \leq \sum\nolimits_{(\tau,c)\in L} Q(\tau, c) \leq \Big\lceil \sum\nolimits_{(\tau,c)\in S} w_c(\tau)\ell(\tau, c) \Big\rceil \ ,$$

which is a strengthening of Condition (C). In addition we maintain $\sum_Q Q(\tau, c)\mu_3(Q) = w_c(\tau)\ell(\tau, c)$.

Suppose that a $w_c(\tau)$-value increases and that $(\tau, c) \in L$ (otherwise we do not need to do anything; observe that if w_{c_r} is increased or decreased by ε the value of $w_{c_r}(\tau_r)\ell(\tau_r, c_r)$ stays constant). We increase $Q(\tau, c)$ for an ε-measure of strategies that have $Q(\tau, c) < 3$. In addition we decrease $Q(\tau_r, c_r)$ for an ε-measure of strategies (only if r is defined, i.e., if $L \neq S$).

Now, there may be an ε-fraction of strategies that has a value of $\sum_{(\tau,c)\in L} Q(\tau, c)$ that

is too large by 1, and there may exist an ε-measure of strategies for which this value is by one too low. As before we can perform re-balancing steps in order to fix this at an expected cost of $\mathcal{O}(\varepsilon)$.

3.3.3. Analysis of the cost control procedure

We now give the analysis of the cost control procedure.

Bounding the violation

The analysis of the base procedure in Section 3.3.2 shows that the $y_c(\tau)$-constraints in the dual solution are only violated by a factor of $W_{\text{base}} = \mathcal{O}(\log \log b)$ when using $\alpha_1(\vec{z}, t)$-variables (Lemma 3.9). Our lower bound construction in Section 3.3.1 uses the $\alpha_2(\vec{s}, t)$-variables. In every iteration of the base procedure, $\alpha_1(\vec{z}, t)$ and $\alpha_2(\vec{s}, t)$ increase by the same amount $\mathrm{d}\alpha$, yet $s_c < z_c$ for active colors c. This means that the α_2-variables generate a higher profit and a larger violation in the $y_c(\tau)$-constraints. We now analyze the violation that is caused by the $\alpha_2(\vec{s}, t)$-variables.

The deactivation procedure (Algorithm 3) is in charge for controlling the violation of the dual program. The procedure makes sure that the following constraint is observed.

deactivation constraint:
The volume collected by a color during a single activity period lies between V and $3V$. Formally, suppose that a color is active during interval $[\alpha_{\text{start}}, \alpha_{\text{end}}]$. Then

$$V \leq \int_{\alpha_{\text{start}}}^{\alpha_{\text{end}}} |E_c(\vec{s}, t)| \, \mathrm{d}\alpha \leq 3V \ .$$

The reason we do not simply deactivate a color once the collected volume reaches the threshold V is the following. A color that is deactivated will usually become marked and obtain a server for the rest of the phase. Moving a server to the color incurs cost. However, if a server is already located at the color, its deactivation is for free. Therefore we call a server *available* for marking when it reaches volume V. If a server moves to the color before its volume reaches $3V$, we immediately deactivate it. This does not incur any additional cost. The deactivation constraint imposes that we may not delay deactivation for too long and we must *mark* a color when its total volume reaches $3V$. This technique will be substantial when analyzing the routine for scheduling $k > 1$ servers in Section 3.4.

The following lemma bounds the violation of the $y_c(\tau)$-constraints in LP Primal.

Lemma 3.17. *The $y_c(\tau)$-constraints in our dual solution (using $\alpha_2(\vec{s},t)$-variables) are violated by at most a factor of $W = W_{\text{base}} + 6V = \mathcal{O}(W_{\text{base}} + V)$.*

Proof. Fix a color c and a time step τ and recall that the $y_c(\tau)$-constraint is

$$\sum_{\vec{v},t\geq\tau} |E_c(\vec{v},\tau)|_t \cdot \alpha(\vec{v},t) \leq 1 \ .$$

The color c can become active at different time steps. We first show that the left-hand side of the constraint only increases during two activity periods of c.

Assume for a contradiction that the color is activated at $t_1 < t_2 < t_3$ and all three activity periods influence the constraint to τ. Let s_1, s_2, and s_3 denote the s_c-values that correspond to t_1, t_2, and t_3, respectively. For $y_c(\tau)$ to be affected by the first activity period we must have $s_1 < \tau \leq t$ for a time step t within the first activity period. Clearly $t < t_2$. Similarly, we get $s_3 < \tau \leq t'$ for some time step t', so $s_3 < \tau < t_2$. However, between consecutive activation points of a color at least $\lceil f_{\max} \rceil$ items of color c arrive, i.e., $\lceil f_{\max} \rceil$ items arrive between time steps t_2 and t_3. When we activate color c at time t_3 we set s_c such that $|E(\vec{s},t_3)| = \lceil f_{\max} \rceil$. This means $t_2 \leq s_3$, which gives a contradiction.

By the deactivation constraint, the left-hand side of a $y_c(t)$-constraint increases by at most $3V$ during a single activity period of c. The total violation collected during any activity period of c is therefore at most $6V$. The violation contributed while c is inactive is

$$\begin{aligned}\int_{\overline{\text{act}_c}} |E_c(\vec{s},\tau)|_t \,\mathrm{d}\alpha &= \int_{\overline{\text{act}_c}} |E_c(\vec{s},\tau)| \,\mathrm{d}\alpha \\ &= \int_{\overline{\text{act}_c}} |E_c(\vec{z},\tau)| \,\mathrm{d}\alpha = \int_{\overline{\text{act}_c}} |E_c(\vec{z},\tau)|_t \,\mathrm{d}\alpha \leq W_{\text{base}} \ .\end{aligned}$$

The equalities follow because for an inactive color c and constraint $y_c(t)$ we have $|E_c(\vec{s},\tau)| = |E_c(\vec{z},\tau)| \leq f_{\max} \leq |E(\vec{z},t)| - b' \leq |E(\vec{s},t)| - b'$, i.e., capping does not have any effect. The total violation is therefore at most $6V + W_{\text{base}}$. □

Fulfilling the constraints

In the following we show that the cost control procedure fulfills the preconditions of Lemma 3.7. We will use that by Line 9 of the deactivation procedure (Algorithm 3), a color is marked upon deactivation if and only if its activity period lies completely inside the current phase.

Claim 3.18. *Marked colors collect at least total volume V in a phase. This gives Condition I.*

Proof. A marked color has an activity period completely inside the phase. It collects at least volume V due to the deactivation constraint. □

Claim 3.19. *Any color collects at most total volume* $6V$ *in a phase. This gives Condition II.*

Proof. A color can be activated at most once in a phase, because after deactivation within the same phase it will become marked. Hence at most two activity periods per color overlap with a phase. The color collects at most volume $3V$ during each of them due to the deactivation constraint. □

The proof that Condition III holds is split into two parts: the volume gap on new colors and the volume gap on stale colors. We first show the former.

Claim 3.20. *The volume gap on new colors is at most* $\sum_{c \in N_i} \Delta_c(I_i) \leq 6\gamma n_i V$*, where* I_i *denotes the time interval of the phase,* N_i *denotes the set of new colors in the phase and* $n_i = |N_i|$*.*

Proof. This follows directly from Claim 3.19 and the fact that the gap is at most a γ-fraction of the total volume (Equation (3.5)). □

For stale colors, the volume gap is 0 if $k = 1$. This is because the phase immediately ends if a new color is marked, so the (only) stale colors always has a server. This gives Condition III for $k = 1$. For $k > 1$, we give an subroutine for scheduling servers on the stale colors in Section 3.4 so that the volume gap is small. This is formalized in the Stale Color Lemma below.

Cost analysis

When analyzing the cost of our algorithm when serving the block requests, we split the cost into two parts. On the one hand there is the *hit cost* which is the cost we incur if none of the k servers is located at the requested color. This is the cost of the block operations scheduled by the base procedure. If the algorithm changes the placement of servers, the corresponding cost is the *movement cost*. Note that these movements may occur even without an explicit block request. Naturally, we can split this cost among the different colors. For example, the movement cost for color c in the phase is the total number of times that we changed the server assignment for color c.

Recall that the cost control procedure observes the scheduling constraint on the non-stale colors.

scheduling constraint:

- a marked color is always assigned a server,
- the remaining colors are only distributed among stale colors.

From this rule it follows that non-stale colors are served by block operations while they are unmarked, and after this they do not incur any cost because they have a server.

Claim 3.21. *The total movement cost on non-stale colors during Phase i is n_i, where n_i is the number of new colors in the phase.*

Proof. We only incur a movement cost if we mark a color. Non-stale colors that become marked are new colors. □

The cost on stale colors depends on our subroutine for scheduling the k servers on the stale colors. We give this algorithm in Section 3.4 which will prove the following lemma.

Lemma 3.22 (Stale Color Lemma)**.** *For $V \geq D$ there is a randomized scheduling algorithm for stale colors that obeys the scheduling and deactivation constraint and guarantees that the volume gap on stale colors in Phase i is at most $3\gamma(1+\gamma)n_iV$. The expected cost on stale colors generated by this algorithm is only $\mathcal{O}\big(V \log k \cdot \mathbb{E}[\sum_i n_i] + \mathrm{cost}_{\mathrm{base}}\big)$.*

Proof for $k = 1$. For $k = 1$, a phase ends as soon as the first color is marked. This means that the (only) stale color always has a server and does not collect a volume gap. The lemma follows immediately. □

3.3.4. Combining the results

We combine the bound on the cost with the lower bound from Corollary 3.8, which gives our main theorem of this chapter.

Theorem 3.23. *There is an algorithm of competitive ratio $\mathcal{O}(\log k(\log k + \log\log b))$ for the generalized reordering buffer management problem.*

Proof. Condition I and Condition II follow from Claim 3.18 and Claim 3.19, respectively. The volume gap on unmarked colors is at most

$$3\gamma(1+\gamma)n_iV + 6\gamma n_iV \leq 10\gamma n_iV \ ,$$

by Claim 3.20 and the Stale Color Lemma, which gives Condition III. We choose $V = D$, which gives that the violation of the LP constraints is at most $W = W_{\mathrm{base}} + 6V =$

$\mathcal{O}(\log\log b) + 6D = \Theta(D)$. Hence, we can use the lower bounds on the optimum cost provided in Corollary 3.8. It remains to analyze the different cost components.

By Lemma 3.14, the fractional assignments to $y_c(t)$-variables in the base procedure contribute at most $2\sum_{\vec{v},t} \alpha_1(\vec{v},t)\big(|E(\vec{v},t)| - b'\big)$. This is at most $2\int_0^{\alpha_{\max}} (|E(\vec{z},t)| - b')\,\mathrm{d}\alpha$ and can be amortized against the lower bound at a loss of a factor of $\mathcal{O}(W) = \mathcal{O}(D)$.

The expected cost of the cost control procedure is $\mathcal{O}\big(V \log k \cdot \mathbb{E}[\sum_i n_i] + \mathrm{cost}_{\mathrm{base}}\big)$ on the stale colors by the Stale Color Lemma. This can be amortized against the cost of the base procedure and the lower bound $\Omega\big(\sum_i n_i\big)$, as this lower bound holds for every partitioning of the request sequence into phases respecting the preconditions. The loss is a factor of $\mathcal{O}(V \log k) = \mathcal{O}(D \log k)$.

The movement cost of the cost control procedure for non-stale colors is $\sum_i n_i$ due to Claim 3.21. This can be amortized against the lower bound $\Omega\big(\sum_i n_i\big)$ at a loss of a factor of $\mathcal{O}(1)$.

Finally, the cost of block requests to non-stale colors is at most $D\sum_c a_c$, where a_c denotes the number of non-stale activity periods for color c. This is due to the fact that a single activity period contains at most D block requests to a color. □

Observe that our proof of the Stale Color Lemma for $k = 1$ already implies that our algorithm is $\mathcal{O}(\log\log b)$-competitive for classical Reordering Buffer Management. This reproves a result by Avigdor-Elgrabli and Rabani [AR13].

3.4. Scheduling k Servers

The previous section shows how our new approach to RBM leads to an optimal $\mathcal{O}(\log\log b)$-competitive algorithm. The goal of this section is to extend this algorithm to the generalized problem of $k > 1$ servers. This is not straightforward: recall that even if there is no buffer ($b = 0$), GRBM reduces to the Paging problem. This observation implies in particular that deterministic algorithms for GRBM will be $\Omega(k)$-competitive.

As we aim for a polylogarithmic competitive ratio, the algorithm we construct is therefore randomized. We will first present the high-level plan of attack we follow. We then define a cleaner model of the subproblem our algorithm has to solve if $k > 1$. Notably, this implies the definition of a specific adversary for the competitive analysis. We then explain the algorithm itself and prove it fulfills the Stale Color Lemma. Finally, we consider the question of designing an optimal $\mathcal{O}(\log\log b + \log k)$-competitive algorithm for GRBM. Our algorithm does not achieve this competitive ratio, hence we explain the limits of our approach and we point towards techniques to overcome them.

3.4.1. Plan of attack

The Stale Color Lemma (Lemma 3.22) imposes essentially that a scheduling algorithm guarantee that the volume gap in Phase i be at most $\mathcal{O}(n_i V)$, while the movement cost is small. When analyzing the volume gap that a randomized scheduling algorithm collects on the stale colors, one is faced with a major obstacle. The collection of volume (and thereby the increase in the volume gap) is to a large extend guided by the increase of α-variables through the base procedure. Unfortunately, the actions of the base procedure depend on $|E(\vec{z}, t)|$, which, in turn, depends on the distribution of servers on the stale colors. This means that we cannot simply treat the increase of volume gap as an input sequence from an oblivious adversary (see Section 1.2.1 for a definition of the adversaries in the field online algorithms). In other words, while the actual input for our algorithm *is* provided by an oblivious adversary, the input for the (artificial) subproblem of scheduling on the stale colors also depends on the random choices we make. This makes the analysis of the randomized scheduling algorithm much more challenging and it requires a careful modeling of the cost and the adversary.

The Marking algorithm for Paging

As mentioned in Section 3.2, our procedure for scheduling multiple servers is inspired by the *Marking* algorithm devised by Fiat et al. for the PAGING problem [FKL+91]. We now give an overview of their algorithm and we sketch the proof of its competitive ratio. See the textbook by Borodin and El-Yaniv for a formal analysis [BE98, Chap. 3].

The PAGING problem is equivalent to GRBM if the buffer has size 0. A sequence of requests arrives online and each request must be processed immediately by one of k servers. Each request has a color and the k servers move on the set of colors. When the request's color c is occupied by some server, it will be processed immediately. Otherwise, one of the servers must move to color c at cost 1. The goal is to minimize the total cost. The *Marking* algorithm partitions the request sequence into phases with the first phase starting with the algorithm. A phase begins with every color unmarked and the marked colors of the previous phase are called *stale*. When a color sees its first request in the current phase, that color is marked and obtains a server for the rest of the phase. This server is chosen uniformly at random from one of the servers residing on the unmarked stale colors. When k colors have been marked, the current phase ends and a new phase starts.

The proof of the competitive ratio is based on two ideas. First, it is not hard to see that an optimal algorithm has cost at least $\sum_i n_i$ over the course of the sequence. Here,

n_i denotes the number of *new* colors that were requested in the current phase, but not the previous one. Second, one can show that the algorithm's expected cost per phase is $\mathcal{O}(\log k)n_i$. To see this, recall that the algorithm only incurs cost by moving a server from an unmarked stale color c' to a marked color c. We charge all of the cost to the n_i new colors and we assume they are marked at the beginning of the phase. A fixed new color c_0 obtains its server from stale color c_1. If c_1 is marked later in the phase, it obtains its server from a different, still unmarked stale color c_2, and so on. As colors $c_1, c_2, \ldots$ are chosen uniformly at random, we expect the number of unmarked stale colors to have halved in the interval between marking colors c_i and c_{i+1}. As there are k stale colors at the beginning of the phase, we expect only $\mathcal{O}(\log k)$ movements to be charged to c_0.

Our approach, while similar at the high level, differs substantially in the details. Aside from the aforementioned issue in defining the adversary, the cost control procedure also changes the definition of marking colors. The marking process depends on deactivation colors, which means that it is influenced mostly by the volume.

Paid requests

Our algorithm requires that we slightly change the definition of a block request in the cost control procedure. The change is mostly technical, yet necessary for adapting the *Marking* algorithm to our scenario. Recall that an active color obtains a new block request whenever $|E_c(\vec{s}, t)| \geq (1+\gamma)|E_c(\vec{s}, t')|$, where t' is the time of the last request. When a color has obtained D requests in a single activity period, no further requests to this color are issued in the current activity period.

For stale colors, we change this algorithm by issuing requests beyond the D-th request in an activity period whenever $|E_c(\vec{s}, t)| \geq (1+\gamma)|E_c(\vec{s}, t')|$. We call the first D requests to a color its *unpaid requests*, all later requests are called *paid requests*. The paid requests can be amortized against the cost of the base procedure, so they do not asymptotically change the cost of our algorithm. We do not require that a block operation is performed on c upon a paid request. In fact, we prove below that the base procedure must have scheduled a block operation on c.

Before we formally prove that the cost of paid requests can, essentially, be ignored, we explain why they are necessary for our strategy. At a high level, our algorithm must distribute $k - n$ servers on k stale colors. In order to minimize the volume gap, we like to have the n *holes* on those colors with smallest rate increase in total volume. However, following the minimum $|E_c(\vec{s}, t)|$ directly results in a high movement cost, as this value can change very often. Instead, we follow the colors with minimum number of requests, as $|E_c(\vec{s}, t)| \approx (1+\gamma)^{r(c)} f_{\max}$, where $r(c)$ is the number of requests to color c. This

(approximate) equality can only hold when we continue to issue requests after the D-th request.

Fact 3.24. *The number of paid requests in a phase is at most* $2\,\mathrm{cost}_{\mathrm{base}}$*, where* $\mathrm{cost}_{\mathrm{base}}$ *is the cost of the base procedure in the phase.*

Proof. Suppose color c receives its j-th request at time t' with $j \geq D$, then $|E_c(\vec{s}, t')| \geq (1+\gamma)^{D-1} f_{\max} \geq (2b+2)/\gamma$. When the next, paid, request for color c is issued, $|E_c(\vec{s}, t)| \geq (1+\gamma)|E_c(\vec{s}, t')|$. This implies that at least $2b+2$ items of color c have arrived in the interval $(t', t]$ and none of them have been removed by block requests to c. As the buffer only holds b items, it follows that the rounding procedure of the base procedure must have scheduled at least one operation on color c in order keep a feasible solution. □

3.4.2. Model of cost and adversary

We now define an abstract framework for analyzing the cost of the algorithm. We also define precisely the power of our adversary, which simplifies the analysis later on.

Cost model

Consider a fixed adversary whose power will be defined later. A *configuration* C of our algorithm consists of a time step t along with the algorithm's decisions up to time t. At any point in time, our algorithm chooses a new configuration based on (i) its current configuration, (ii) the adversary's actions, and (iii) some random bits. Given that the adversary is fixed, the algorithm's decision is therefore memoryless (more precisely, all previous configurations are encoded in the current one). The algorithm can therefore be viewed as a *Markov chain* whose states are the configurations. Furthermore, this Markov chain is a directed tree rooted at the initial configuration C_0. There exists an edge from configuration C to configuration C' if the algorithm, when in configuration C, decides with non-zero probability to have C' as its next configuration. Let $p(C \to C')$ denote this *transition probability.*

The transition probability naturally extends to arbitrary pairs of configurations as follows: $p(C \to C')$ is 0 if no path from C to C' exists in the tree; otherwise, it is the product of transition probabilities on that path. We say that C' is *reachable* from C, writing $C \to C'$, if and only if $p(C \to C') > 0$. Similarly, when transitioning from configuration C to configuration C', the *cost* of the transition is the total number of unpaid actions performed by the algorithm while transitioning, either as block operations or server movements on the stale colors. If $C \to C'$, let $\mathrm{cost}(C \to C')$ denote the cost

of the transition from C to C'; otherwise $\text{cost}(C \to C') = 0$. Importantly, $\text{cost}(C \to C')$ ignores the cost that happens right after a paid request. We show later that each paid request incurs only constant cost, so the total cost of paid requests is $\mathcal{O}(\text{cost}_{\text{base}} + n(C)D)$ in a phase ending in configuration C.

Our analysis uses an amortization scheme for the cost of the algorithm. For any nonnegative function $\Phi(C)$ from the set of configurations to the real numbers, let the *amortized* cost of transitioning from C to C' be

$$\text{cost}_\Phi(C \to C') = \begin{cases} \text{cost}(C \to C') + \Phi(C') - \Phi(C) & \text{if } C \to C' \ , \\ 0 & \text{otherwise.} \end{cases} \ . \tag{3.16}$$

Fact 3.25. *If $C \to X$ and $X \to C'$ then*

$$\begin{aligned} p(C \to C') &= p(C \to X) \cdot p(X \to C') \ , \\ \text{cost}_\Phi(C \to C') &= \text{cost}_\Phi(C \to X) + \text{cost}_\Phi(X \to C') \ . \end{aligned}$$

Proof. If $C \to X$ and $X \to C'$ then X lies on the path from C to C' in the tree. Thus both the sides of the equation equal the product of transition probabilities on that path. Similarly,

$$\begin{aligned} \text{cost}_\Phi(C \to C') &= \text{cost}(C \to C') + \Phi(C') - \Phi(C) \\ &= \text{cost}(C \to X) + \text{cost}(X \to C') + \Phi(C') - \Phi(X) + \Phi(X) - \Phi(C) \\ &= \text{cost}_\Phi(C \to X) + \text{cost}_\Phi(X \to C') \ . \end{aligned}$$

□

The expected cost of our algorithm on the stale colors is

$$\sum_{B \in \mathcal{B}} p(C_0 \to B)\, \text{cost}(C_0 \to B) \ .$$

In order to show that this is small, our strategy is to define a special set $\mathcal{S}$ of *event configurations*. We show that our algorithm only encounters few event configurations in phase, yet the expected cost of transitioning from one event configuration to the next is small. For event configurations $C, C' \in \mathcal{S}$, we write $C \mapsto C'$ (say C' is *successor* of C) if $C \mapsto C'$ and there is no event configuration $X \in \mathcal{S}$ on the path from C to C'. Let $p(C \mapsto C') = p(C \to C')$ if $C \mapsto C'$ and 0 otherwise. Similarly, let $\text{cost}_\Phi(C \mapsto C') = \text{cost}_\Phi(C \to C')$ if $C \mapsto C'$ and 0 otherwise.

The precise definition of event configurations is deferred to Section 3.4.3. For now, we

only require the event configurations and a suitable potential function $\Phi \geq 0$ to fulfill the following properties. Let $n(C)$ denote the number of new colors marked in the phase of C up to the time step of C.

(P1) The algorithm experiences at most $\mathcal{O}(D \log k)$ event configurations per phase.

(P2) For any $C \in \mathcal{S}$, $\sum_{C' \in \mathcal{S}} p(C \mapsto C') \operatorname{cost}_\Phi(C \mapsto C') = \mathcal{O}(n(C))$.

(P3) $\Phi(C_0) = 0$ and $\Phi(C) \geq 0$ for every configuration C.

Using these properties, the following technical lemma gives an upper bound on the algorithm's expected cost on stale colors.

Lemma 3.26. *The expected cost on stale colors is at most* $\mathcal{O}(D \log k)\mathbb{E}\left[\sum_i n_i\right]$, *where* n_i *is the number of new colors in phase* i.

Proof. Without loss of generality, assume that the end of a phase is always an event configuration. Let $\mathcal{P}$ be the set of the event configurations that end a phase and call them *phase configurations*. For any phase configuration, we have define an artificial, unique event configuration on the next time step that marks the start of the new phase. Similarly to the event configurations, we write $C \overset{\mathfrak{p}}{\mapsto} C'$ if $C \to C'$ and there is no $X \in \mathcal{P}$ with $C \to X$ and $X \to C'$. Let $p(C \overset{\mathfrak{p}}{\mapsto} C') = p(C \to C')$ if $C \overset{\mathfrak{p}}{\mapsto} C'$ and 0 otherwise.

The expected cost of our algorithm is, using Property (P3) and Equation (3.16),

$$\begin{aligned}\sum_{B \in \mathcal{B}} p(C_0 \to B) \operatorname{cost}(C_0 \to B) &= \sum_{B \in \mathcal{B}} p(C_0 \to B)(\operatorname{cost}_\Phi(C_0 \to B) - \Phi(B) + \Phi(C_0)) \\ &\leq \sum_{B \in \mathcal{B}} p(C_0 \to B) \operatorname{cost}_\Phi(C_0 \to B) \ .\end{aligned}$$

In a first step, we show that this is at most $\mathcal{O}(D \log k) \sum_{P \in \mathcal{P}} p(C_0 \to P) n(P)$. We then prove that $\sum_{P \in \mathcal{P}} p(C_0 \to P) n(P)$ equals $\mathbb{E}\left[\sum_i n_i\right]$, which gives the lemma.

Claim 3.27. $\sum_{B \in \mathcal{B}} p(C_0 \to B) \operatorname{cost}_\Phi(C_0 \to B) = \mathcal{O}(D \log k) \sum_{P \in \mathcal{P}} p(C_0 \to P) n(P)$.

Proof. Fact 3.25 implies that for any $B \in \mathcal{B}$, $\operatorname{cost}(C_0 \to B)$ can be written as the sum of costs between the event configurations on the path from C_0 to B in the tree. We use that $\operatorname{cost}(C \mapsto C')$ is 0 if C' is not a successor of C and Property (P3) which gives

$$\operatorname{cost}_\Phi(C_0 \to B) \leq \sum_{C \in \mathcal{S}} \sum_{\substack{C' \in \mathcal{S} \\ C' \to B}} \operatorname{cost}_\Phi(C \mapsto C') \ .$$

This allows us to rewrite the expected cost of the algorithm as

$$\begin{aligned}
&\sum_{B\in\mathcal{B}} p(C_0 \to B) \sum_{C\in\mathcal{S}} \sum_{\substack{C'\in\mathcal{S}\\ C'\to B}} \operatorname{cost}_\Phi(C \mapsto C') \\
= &\sum_{C\in\mathcal{S}} \sum_{B\in\mathcal{B}} \sum_{\substack{C'\in\mathcal{S}\\ C'\to B}} p(C_0 \to B) \operatorname{cost}_\Phi(C \mapsto C') \\
= &\sum_{C\in\mathcal{S}} p(C_0 \to C) \sum_{B\in\mathcal{B}} \sum_{\substack{C'\in\mathcal{S}\\ C'\to B}} p(C \to B) \operatorname{cost}_\Phi(C \mapsto C') \\
= &\sum_{C\in\mathcal{S}} p(C_0 \to C) \sum_{C'\in\mathcal{S}} \operatorname{cost}_\Phi(C \mapsto C') \sum_{\substack{B\in\mathcal{B}\\ C'\to B}} p(C \to B) \ .
\end{aligned}$$

The second equation used Fact 3.25, the third equation changed the order of summation. If C' is reachable from C, Fact 3.25 gives $\sum_{B\in\mathcal{B},C'\to B} p(C \to B) = p(C \to C')$ as the algorithm always ends in a configuration of $\mathcal{B}$. The expected cost is therefore

$$\sum_{C\in\mathcal{S}} p(C_0 \to C) \sum_{C'\in\mathcal{S}} p(C \to C') \operatorname{cost}_\Phi(C \mapsto C') \le \zeta \cdot \sum_{C\in\mathcal{S}} p(C_0 \to C) n(C) \ .$$

We used Property (P2) with ζ as the constant from the $\mathcal{O}$-notation.
We now multiply each term of the sum with $1 = \sum_{P\in\mathcal{P}} p(C \overset{\mathfrak{p}}{\mapsto} P)$ and obtain

$$\begin{aligned}
\zeta \cdot \sum_{C\in\mathcal{S}} p(C_0 \to C) n(C) \sum_{P\in\mathcal{P}} p(C \overset{\mathfrak{p}}{\mapsto} P) &= \zeta \cdot \sum_{P\in\mathcal{P}} \sum_{C\in\mathcal{S}} p(C_0 \to C) n(C) p(C \overset{\mathfrak{p}}{\mapsto} P) \\
&\le \zeta \cdot \sum_{P\in\mathcal{P}} n(P) \sum_{C\in\mathcal{S}} p(C_0 \to C) p(C \overset{\mathfrak{p}}{\mapsto} P) \\
&\le \zeta' \cdot D \log k \sum_{P\in\mathcal{P}} p(C_0 \to P) n(P) \ .
\end{aligned}$$

The first inequality uses that the number of colors never decreases within a phase. To see the second inequality, observe that Property (P1) implies that for any fixed phase configuration P at most $\mathcal{O}(D \log k)$ event configurations have positive $p(C \overset{\mathfrak{p}}{\mapsto} P)$. Again ζ' is a constant from the $\mathcal{O}$-notation. □

Claim 3.28. $\sum_{P\in\mathcal{P}} p(C_0 \to P) n(P) = \mathbb{E}\left[\sum_i n_i\right]$.

Proof. Let $\mathcal{P}_i$ denote the phase configurations ending the i-th phase in the tree and observe that the $\mathcal{P}_i$ partition $\mathcal{P}$. The expected number $\mathbb{E}[n_i]$ of new colors in Phase i is $\sum_{P\in\mathcal{P}_i} p(C_0 \to P)$. This uses that the algorithm has 0 colors in any phase it does not reach. The claim then follows by linearity of expectation. □

□

Model of the adversary

From now on, we consider a single phase starting in a single configuration. We re-number the time steps with $t = 1$ being the first time step of the phase. The request sequence and the increase in α both depend on the color sequence fed to the base procedure; it is convenient to model it as a specific type of adversary that we define next.

We assume that the adversary select an order among the block requests on the stale colors. Below, we show that any request outside this sequence may be ignored. After choosing this sequence, the adversary can do one of the following operations in each Step t.

- **Request:** Issue the next request from the predefined sequence.
- **Volume increase:** Increase α by $\mathrm{d}\alpha$. This increases the volume gap on active colors that are not assigned a server.

In addition there are the following events that the adversary cannot influence directly but only indirectly by altering the total volume collected by a color.

- **Marking of a stale color:** In this case one server has to be moved to the newly marked stale color.
- **Marking of a new color:** Mark a non-stale color c. In this case one server has to be moved away from the stale colors to be assigned to color c for the remainder of the phase. This increases the number $n(t)$ of new colors that appeared in the phase up to time t.

For each of the above actions the adversary *knows the random choices of the algorithm up to Step t*. The following claim shows that we can indeed assume that the adversary has to specify the order of requests at the start of the phase.

Claim 3.29. *After the start of the phase the order of requests to unmarked stale colors during the phase is not influenced by actions of the scheduling algorithm.*

Proof. As a reminder the rules for generating requests are as follows:

1. an inactive color gets a request if it has seen at least $f_{\max}$ requests since the most recent explicit server request

2. an active color gets a request if $|E_c(\vec{s}, t)| \geq (1 + \gamma)|E_c(\vec{s}, t')|$, where t' is the time step of the most recent explicit request to the color.

Recall that stale colors are inactive at the beginning of the phase. If at the start of the phase we know the number of items that arrived for every color since the most recent request, we can calculate how many items need to arrive in order for a request of Type 1 to be generated (which activates the color). As long as the color stays active further requests of Type 2 only depend on the number of items received for the respective color. When the color gets deactivated it becomes marked and, hence, further requests to this color are not of interest. This shows that requests to unmarked stale colors only depend on the number of items and, thus, their order cannot be influenced once the input sequence of items is fixed, which has to be done at the start of the algorithm. □

At first glance the above claim might appear too weak to guarantee that the adversary is strong enough as the order of requests to marked colors *can* actually be influenced by actions of the base procedure. The reason is that marking/deactivating a color changes how requests to the color are generated, and thus this can influence order of requests. However, our algorithm ensures that any marked color is always assigned a server. It follows that adding or removing requests to marked colors does not influence the scheduling decisions or the cost of the algorithm.

In order to get a cleaner model we restrict the choice of the online algorithm in each step. We add (many) dummy time steps, where in each time step the adversary has the choice between a limited number of actions. More concretely we have the following types of steps:

- *Request time steps* $t \in \mathcal{T}_r$
  The adversary issues the next request in the predetermined sequence. The adversary *cannot* skip this step.

- *Marking time steps* $t \in \mathcal{T}_n$ for new colors
  The adversary increases n or skips the action.

- *Volume time steps* $t \in \mathcal{T}_v$
  The adversary either increases α by $\mathrm{d}\alpha$ or skips the action.

The above model can be obtained by fixing the time steps that should be request time steps and by adding sufficiently many time steps of each type between any two consecutive request time steps.

3.4.3. Subroutine for scheduling servers

We now present our algorithm and its analysis. Observe that with the discussion of the previous section, it suffices to consider the algorithm's decisions in a single phase.

The algorithm

Given a configuration C and stale color c, let $r_c(C)$ be the number of requests to color c up to the time step of C. Let $\ell_r(C)$ denote the number of unmarked stale colors that have seen r requests. The current *round* $r^*(C)$ is $\min\{r \mid \ell_r(C) \geq n(C)\}$. We sometimes omit the argument C if the configuration is clear. Observe that r^* never decreases as ℓ_r never increases and n never decreases. A color c is *free* in round r^* if it is an unmarked color having seen exactly r^* requests and with a server located at it.

The algorithm's movement strategy is the following. The current server distribution is only modified if

- a color c without server obtains its (r^*+1)-th requests, or
- a new color c is marked, or
- a stale color c without server is marked.

In these cases, the algorithm picks a free color c' uniformly at random and moves the server from c' to c.

Similar to classical *Marking*, our algorithm distributes the servers on the marked colors and the stale colors that have seen the most requests.

Observation 3.30. *A marked color has a server. A stale color with more than r^* requests also has a server. An unmarked stale color with less than r^* requests has no server.*

Proof. A server moves to a color when the color is marked. As marked colors never become free, this server never moves away.

Fix a configuration C and let t be its time step. For $0 \leq \tau \leq t$, we write $r_c(\tau)$ to denote the number of requests to color c up to time τ and $r^*(\tau)$ to denote the current round at time τ. Note that configuration C encodes all decisions up to time t, hence $r_c(\tau)$ and $r^*(\tau)$ are fixed for $0 \leq \tau \leq t$. Let c be some unmarked stale color.

Suppose $r_c(t)$ is larger than $r^*(t)$. We have $r_c(0) = r^*(0) = 0$, as a phase starts without marked colors. Let $t' < t$ be the last time step when $r_c(t') = r^*(t')$. As both r^* and r_c are nondecreasing, it follows that $r_c(t'+1) = r^*(t'+1) + 1$ and $r_c(\tau) > r^*(\tau)$ for $t'+1 \leq \tau \leq t$. On the one hand, this implies that either a server was located at c at time

$t' + 1$, or a server moved to c at time $t' + 1$. On the other hand, c was no free color in the interval $[t' + 1, t]$, so the server cannot have moved away. Hence there is a server at color c.

Now suppose $r_c(t)$ is smaller than $r^*(t)$. Let $t' < t$ be the last time during which $r_c(t') = r^*(t')$. It follows that r^* increases at time $t' + 1$, so $\ell_{r^*} = n$ at time t'. We know that every color with at least $r^* + 1$ requests has a server and n stale colors have no server. It follows that no unmarked stale color with at most $r^*(t')$ requests has a server at time t'. In particular, there is no server at color c at time t'. As color c is unmarked and never sees its $(r^* + 1)$-th request during interval $[t' + 1, t]$, it still has no server at time t. □

It follows that our algorithm fulfills the scheduling constraint. It also fulfills the deactivation constraint as marked colors always have a server and do not collect any more volume.

Bounding the volume gap

In order to obtain a bound on the volume gap, we proceed as follows. The volume gap on an active color c over interval I is defined as

$$\Delta_c(I) = \mathrm{vol}_c^T(I) - \mathrm{vol}_c^E(I) \leq \gamma \mathrm{vol}_c^T(I) \ .$$

The inequality follows from Equation (3.5). Observe that a color that is assigned a server does not increase its volume gap as the increase in total volume matches the increase in extra volume. Instead of analyzing the volume gap directly, we instead consider the *volume cost*, which is equal to the total volume a color without server collects.

Lemma 3.31. *The volume cost on stale colors is at most* $3(1 + \gamma)V \cdot n$ *per phase, where* n *is the number of new colors of the phase.*

Proof. Fix a configuration C that marks the end of a phase and let $n = n(C)$. We show that the volume cost in the phase ending in configuration C is $3(1 + \gamma)V \cdot n$. Note that this imposes a fixed set of random choices for the phase.

If $n = 0$ a stale color is always assigned a server, so the volume cost on stale colors will be 0 in this case. Hence, assume $n > 0$. In this case there exist n stale colors that are not marked in the phase. Let U denote this set of colors and observe that each of these colors collects at most total volume $3V$ during the phase. Otherwise, the color would be deactivated and directly become marked (unless the phase ends at that time step). Hence $\sum_{c \in U} \mathrm{vol}_c^T(I) \leq 3nV$, where I is the interval of the phase. We show next that in

each time step the volume cost of the algorithm increases roughly by the total volume collected by colors in U during the step.

Fix a time step and let M denote the set of active stale colors that are currently missing a server. The volume cost increases by $\sum_{c\in M} |E_c(\vec{s},t)|\,\mathrm{d}\alpha$. On the other hand the total volume collected by colors in U increases by $\sum_{c\in U} |E_c(\vec{s},t)|\,\mathrm{d}\alpha$.

Claim 3.32. $\sum_{c\in M} |E_c(\vec{s},t)| \le (1+\gamma)\sum_{c\in U} |E_c(\vec{s},t)|$.

Proof. First observe that $|M| \le n = |U|$. We sort the colors in M and U according to the number of block requests they have received so far.

We now map the i-th color in M (if it exists) to the i-th color in U. We show that for any such pair (c_m, c_u) we have $|E_{c_m}(\vec{s},t)| \le (1+\gamma)|E_{c_u}(\vec{s},t)|$.

Let $r(c)$ denote the number of requests to color c so far. Clearly $r(c_m) \le r(c_u)$ as only the colors that have seen the fewest requests are missing a server by Observation 3.30. Furthermore, $r(c_m) \ge 1$, as the colors of M are active and therefore must have been requested at least once. Let ρ_i denote the number of items in $E_c(\vec{s},t)$ right after the i-th request, for $1 \le i \le D$. Due to the way requests are generated in Algorithm 2 there are exactly $\lceil f_{\max} \rceil$ items in $E_c(\vec{s},t)$ at the time of the first request. For $i > 1$, we have $s_i = \lceil (1+\gamma)s_{i-1} \rceil$. Hence if $1 \le r(c_m)$,

$$\begin{aligned} |E_{c_m}(\vec{s},t)| &\le s_{r(c_m)+1} - 1 \\ &\le \lceil (1+\gamma)s_{r(c_m)} \rceil - 1 \\ &\le (1+\gamma)s_{r(c_u)} \le (1+\gamma)|E_{c_u}(\vec{s},t)| \ . \end{aligned}$$

□

Since the volume of elements in U is at most $3nV$ we get that the volume cost on stale colors is at most $3(1+\gamma)nV$. □

Bounding the cost

In order to bound the movement cost of the servers, we define a suitable set of event configurations. The start of a phase is always an event configuration, as well as the configuration when $n(C)$ first increases to 1. After that, event configurations are defined inductively. As long as $r^* < D$, a configuration is an event configuration if either $n(C)$ has doubled since the last event configuration, or the number of free colors has decreased by a factor of 1/4, or the round $r^* < D$ increases. Furthermore, the end of a phase (which are just one artificial time step away from the beginning of the next) is an event

configuration. When $r^* \geq D$, no further more event configurations exist before the end of the phase.

We prove that this definition fulfills properties (P1), (P2), and (P3) for a suitable potential function Φ. The next claim proves Property (P1).

Claim 3.33. *There are at most $8D \log k$ event configurations in a phase.*

Proof. The number of new colors is at most k, hence it doubles at most $\log k$ times. At most D event configurations mark the increase of r^*. Within a round, the number of free colors decreases by a factor of at least $3/4$ between event configurations, hence there are at most $\log_{4/3} k < 4 \log k - 1$ such event configurations per round. Finally, there are two event configurations marking the start and end of the round. Summing up (generously) gives the claim. □

The movement cost for marking stale colors is bounded via the following technique. According to the deactivation constraint, a color that as collected total volume V is *available* for marking. The color *must* be marked before it collects volume $3V$. If a color is available for marking and has a server assigned to it, we simply mark the color. This does not induce any movement cost, as the server assignment does not change. If a color reaches volume $3V$ without a server assigned to it, we mark the color and reassign a server from a free color to it. This *forced marking* induces cost 1. The next claim bounds the number of these forced marking operations.

Claim 3.34. *For any event configuration C, there can be at most $4n(C)$ forced marking operations before the next configuration.*

Proof. Observe that a color that experiences a forced marking operation has collected total volume $2V$ since it became available for marking. During this time it has never been assigned a server; otherwise, it would have been marked right away. This means that a color that experiences a forced marking operation has volume cost at least $2V$.

Let configuration C' be a successor of C and assume without loss of generality that C ends a phase. This can be achieved, e.g., by marking all stale colors without server right before C'. Then the volume cost collected in the phase ending with C' is at most $3(1+\gamma)V \cdot n(C') \leq 6(1+\gamma)V \cdot n(C)$ by Lemma 3.31. The number of forced markings in the phase is therefore at most

$$\frac{6(1+\gamma)V \cdot n(C)}{2V} = 3(1+\gamma)n(C) \leq 4n(C). \qquad \square$$

Next, we show that the cost after round $r^* = D$ is small.

Claim 3.35. *The cost after round D is at most $\mathcal{O}(n(C)D + \text{cost}_{\text{base}})$ if the phase ends in configuration C.*

Proof. After reaching round D, at most $n(C)D$ requests are still issued for colors with less than D requests so far. All requests after the D-th request to a color is a paid request and the algorithm has constant cost for adjusting the distribution in response to it. The total number of paid requests is $\mathcal{O}(\text{cost}_{\text{base}})$, which gives the claim. □

We now prove Property (P2). Let

$$\Phi(C) = \sum_c \max\{0, \min\{r^*(C), D\} - r_c(C)\} \ .$$

Clearly $\Phi(C) \geq 0$ and $\Phi(C_0) = 0$, so Property (P3) is fulfilled.

Lemma 3.36. *For any $C \in \mathcal{S}$, $\sum_{C' \in \mathcal{S}} p(C \mapsto C') \, \text{cost}_\Phi(C \mapsto C') = \mathcal{O}(n(C))$.*

Proof. If $r^*(C) \geq D$, all requests before the next event configuration are paid requests, so $\text{cost}_\Phi(C \mapsto C') = 0$. As Φ only decreases after round D, the lemma holds.

Partition the set of successors of C into sets $\mathcal{S}_j$ so that the adversary makes identical decisions for all configurations in $\mathcal{S}_j$. We fix one set $\mathcal{S}_j$ and show that the expected cost before reaching a configuration in $\mathcal{S}_j$ is $\mathcal{O}(n(C))$.

Instead of arguing about the cost of servers, it is convenient to consider the cost of *antiservers* that mark the absence of a server. We imagine n antiservers moving on the stale colors, each color having either a server or an antiserver. There are n antiservers, with $n(C) \leq n \leq 2n(C)$, as there is one antiserver per new color and the number of new colors can at most double between event configurations. The antiservers only visit free colors (with $r_c = r^*$) and unmarked colors with $r_c < r^*$ by Observation 3.30.

When a color c without server and $r_c < r^*$ is requested, the unit cost for processing the request is absorbed by the decrease in Φ. If the requested color c is a free color, so $r_c = r^*$, we charge the antiserver located at c with 1. We also charge an antiserver located at a color that is marked. Let A_i be the cost charged to the i-th antiserver and let M_i be the number of times its color is marked in $\mathcal{S}_j$. We show that $\mathbb{E}[A_i \mid M_i = m_i] = \mathcal{O}(1 + m_i)$.

As the request sequence cannot be modified by the adversary after the start of the phase, we number the free colors in the order in which they obtain their $(r^* + 1)$-th request. If configuration C has z free colors, at most colors $1, \ldots, z/4$ are requested before the next event configuration. The adversary only controls the marking operations; we therefore consider some fixed set of marking operations.

The m_i forced markings of colors with antiserver i partition the time interval into $1 + m_i$ intervals during which the antiserver experiences no marking event. Consider any of these

intervals. Whenever antiserver i is charged, this means that is was located on one of the colors $1, \dots, z/4$ and this color just received its $(r^* + 1)$-th request. The antiserver then picks a free color uniformly at random to move to. If the new color is one of $z/4+1, \dots, z$, the antiserver is *settled*, as it will not be charged before the next marking event. As there are always at least $z/2$ unmarked colors in the interval $z/4 + 1, \dots, z$, it follows that antiserver i is settled with probability at least $2/3$. Let B_i count the number of trials until antiserver i is settled. It follows that $B_i < G$ with $G \sim \text{Geo}(2/3)$, so $\mathbb{E}[B_i] < \mathbb{E}[G] = 3/2$. With the above discussion, this implies that $\mathbb{E}[A_i \mid M_i = m_i] \leq \frac{3}{2}(1 + m_i) + m_i$.

It follows that for any set $\mathcal{S}_j$, the expected cost is at most

$$\sum_i \mathbb{E}[A_i \mid M_i = m_i] \leq 2n(C)\frac{3}{2} + \frac{5}{2}\sum_i m_i = 3n(C) + 4n(C) = 7n(C) \ .$$

We have used that by Claim 3.34 the number of forced markings between event configurations is at most $4n(C)$. We also used that there are at most $2n(C)$ antiservers.

As the expected cost for any fixed $\mathcal{S}_j$ is at most $7n(C)$, the same must hold for any weighted average of successors of C. □

Combining the results

Lemma 3.36 and Claim 3.33 imply that our event configurations and Φ fulfill the required properties, so Lemma 3.26 and Claim 3.35 imply that the expected cost on the stale colors is at most $\mathcal{O}\big(D \log k \cdot \mathbb{E}[\sum_i n_i] + \text{cost}_{\text{base}}\big)$. The bound on the volume gap is shown in Lemma 3.31. Hence, we have proven the Stale Color Lemma for $k > 1$.

Lemma 3.37 (Stale Color Lemma)**.** *For $V \geq D$ there is a randomized scheduling algorithm for stale colors that obeys the scheduling and deactivation constraint and guarantees that the volume gap on stale colors in Phase i is at most $3\gamma(1+\gamma)n_iV$. The expected cost on stale colors generated by this algorithm is only $\mathcal{O}\big(V \log k \cdot \mathbb{E}[\sum_i n_i] + \text{cost}_{\text{base}}\big)$.*

As shown in Section 3.3.4, this implies our main theorem of this chapter.

Theorem 3.38. *There is a $\mathcal{O}(\log k(\log k + \log\log b))$-competitive algorithm for Generalized Reordering Buffer Management.*

3.4.4. Towards optimal competitiveness?

Improving the competitive ratio in Theorem 3.23 to $\mathcal{O}(\log k + \log\log b)$ is a challenge we must leave open in this work. Indeed, while a lower bound of $\Omega(\log k + \log\log b)$ is evident from the connection to Paging and RBM, we are not aware of any other result

to bridge this gap. Our goal for the remainder of the section will be to highlight some possible avenues for bridging this gap within our framework, as well as the difficulties that are to overcome.

First, observe that in order to achieve optimal competitiveness, it is enough to give an algorithm for scheduling on the stale colors that respects the volume gap, and whose total cost on the stale colors $\mathcal{O}(V + \log k) = \mathcal{O}(V)$. Recall that the algorithm we propose only achieves cost in $\mathcal{O}(V \cdot \log k)$.

In order to gain a better intuition for the problem, we propose a discussion on the following simplified problem we call V-paging. In this problem, we are given a cache of size $k - n$ and k colors (pages). At each time step, one of the k colors is requested and must be moved into the cache at cost 1. After a color has been requested V times, it is marked and stays in the cache permanently. The request sequence ends after $k - n$ colors have been marked. The goal is to find a strategy to move colors in and out of the cache whose cost is minimal.

Strategies for V-Paging

A simple strategy for the offline version of the problem leaves those n colors that will never be marked outside of the cache permanently. This strategy incurs movement cost n and hit cost at most $n \cdot V$. Observe that any deterministic online strategy tie breaking cannot avoid cost $\Omega(V(k - n))$, as the adversary can essentially request a color that is outside of the algorithm's cache most of the time.

A the high level, a randomized strategy for V-Paging can be defined as follows. At a time step t, let $x_c(t)$ denote the number of requests color c has received before time step t. This gives a vector x over the colors that we call the *load vector*. Similarly, let $\ell_c(t) = 1$ if color c is requested at time t and $\ell_c(t) = 0$ otherwise. This gives $x(t) = x(t-1) + \ell(t-1)$. A randomized algorithm must define for any time step the probability $\rho_c(x, t)$ that color c is *not* in the cache. Observe that the entries of vector ρ must sum to n at any time.

The algorithm we present in Section 3.4.3 chooses $\rho_c = 1$ for colors with minimum value of x_c, breaking ties randomly. It is therefore a *Follow-the-leader* strategy and we saw above that it incurs hit cost $n \cdot V \log k$.

An improved strategy comes from the area of online learning and is given by Blum et al. [BBK99]. They propose to adapt the *Randomized Weighted Majority algorithm* (RWM) for paging. This algorithm is known under many different names such as *Multiplicative Weights Update* or *Regularized Follow-the-leader*. We sketch their approach using a different notation.

For $x \in \mathbb{R}^k$, positive integer $n \leq k$, and $\beta > 1$, define the β-softmin of order n as

$$\operatorname{smin}_{\beta,n}(x) = -\log_\beta\left(\sum_{T\in\binom{[k]}{n}} \beta^{-\sum_{i\in T} x_i}\right) .$$

The function $\operatorname{smin}_{\beta,n}(x)$ is a smooth approximation of the sum of the n smallest elements from the vector x. The higher we choose factor β, the more tight is the approximation of the softmin to the actual minimum. We show in Section A.2 of the appendix that the β-softmin fulfills the following properties.

Claim 3.39. *The k-dimensional β-softmin of order $n \geq 1$ fulfills the following properties:*

(a) The function approximates the sum of the n smallest entries in x. Formally for all vectors x we have

$$\min_{T\in\binom{[k]}{n}} \sum_{i\in T} x_i - n\log_\beta k \leq \operatorname{smin}_{\beta,n}(x) \leq \min_{T\in\binom{[k]}{n}} \sum_{i\in T} x_i .$$

(b) The gradient of the function is the type of measure we require. Formally, for all vectors x, $0 \leq \nabla\operatorname{smin}_{\beta,n}(x)_c \leq 1$, and $\sum_c \nabla\operatorname{smin}_c(x)_c = n$.

(c) The change of the function is reasonably well approximated by its gradient. Formally for all vectors x, ℓ, if $\|\ell\|_1 \leq 1$ then

$$\operatorname{smin}_{\beta,n}(x+\ell) - \operatorname{smin}_{\beta,n}(x) \geq \frac{1-1/\beta}{\ln\beta}\nabla\operatorname{smin}_{\beta,n}(x)^T\ell .$$

Algorithm RWM uses measure $\rho(x(t),t) = \nabla\operatorname{smin}_{\beta,n}(x(t))$ with constant β . The expected hit cost at time step t is therefore $\nabla\operatorname{smin}_{\beta,n}(x(t))_c = \nabla\operatorname{smin}_{\beta,n}(x(t))^T\ell(t)$, if color c is requested at that time step. This gives the following lemma.

Lemma 3.40 ([BBK99])**.** *The expected hit cost of RWM is $\mathcal{O}(n(V+\log_\beta k))$ for constant $\beta > 1$.*

Proof. By Claim 3.39(c), the total hit cost is

$$\begin{aligned}\sum_{t\geq 1} \nabla\operatorname{smin}_{\beta,n}(x(t))^T \cdot \ell(t) &\leq \frac{\ln\beta}{1-\beta^{-1}} \sum_{t\geq 1}\Big(\operatorname{smin}_{\beta,n}(x(t)+\ell(t)) - \operatorname{smin}_{\beta,n}(x(t))\Big) \\ &= \frac{\ln\beta}{1-\beta^{-1}}\Big(\operatorname{smin}_{\beta,n}(x(T)) - \operatorname{smin}_{\beta,n}(x(1))\Big) .\end{aligned}$$

Here T denotes the time step after the last request. Claim 3.39(a) gives that $\text{smin}_{\beta,n}(x(T))$ is at most nV, as there exist n colors that see at most V requests. On the other hand, $x(1)$ is 0 in ever component, so $\text{smin}_{\beta,1}(x(1)) \geq n \log_\beta k$. □

Blum et al. furthermore give a strategy to round measure ρ into a distribution over valid cache state so that the probability of color c being outside the cache is indeed ρ_c. They show that their strategy's expected movement cost is at most twice the hit cost. We refer to their paper for the details [BBK99]. The authors prove that the total cost is at most $\mathcal{O}(n(V + \ln k))$.

In order to obtain a lower bound on the performance of RWM, consider requesting the colors in round-robin fashion. We show that the expected hit cost of RWM degrades with β.

Lemma 3.41. *For $k, \beta > 2$, the expected hit cost of RWM is at least*

$$V \cdot \min\{\ln(\beta), \ln(k)\}/2 \ .$$

Proof. Let $n = 1$ and fix a round r of the request sequence. Right before the request to page i for $i \in [k]$, pages $1, \ldots, i-1$ have been requested r times, and color $i, \ldots, k$ have been requested $r - 1$ times. It follows that the probability mass on color i is

$$\nabla \text{smin}_{\beta,1}(x)_i = \frac{\beta^{-x_i}}{\sum_j \beta^{-x_j}} = \frac{1}{(i-1)\frac{1}{\beta} + (k-i+1)} \ .$$

The second equation follows by dividing by β^{-r+1}. The expected hit cost in round r is therefore

$$\begin{aligned} \sum_{i=1}^{k} \frac{1}{(i-1)\frac{1}{\beta} + (k-i+1)} &= \sum_{i=1}^{k} \frac{1}{i(\frac{1}{\beta} - 1) + (k+1-\frac{1}{\beta})} \\ &= \frac{\beta}{1-\beta} \sum_{i=1}^{k} \frac{1}{i+\zeta} \ , \end{aligned}$$

with $\zeta = \frac{k+1-1/\beta}{1/\beta-1}$. The sum is at least $\int_1^k \frac{1}{x+\zeta}\,\mathrm{d}x$, so the expected hit cost is at least

$$\begin{aligned}\frac{\beta}{1-\beta}\ln\left(\frac{k+\zeta}{1+\zeta}\right) &= \frac{\beta}{\beta-1}\ln\left(\frac{1+\zeta}{k+\zeta}\right)\\ &= \frac{\beta}{\beta-1}\ln\left(\frac{k}{\frac{k}{\beta}+1-\frac{1}{\beta}}\right)\\ &= \frac{\beta}{\beta-1}\ln\left(\frac{\beta k}{k+\beta-1}\right)\ ,\end{aligned}$$

As $\beta > 2$, this is at least $\ln(\beta k/(k+\beta-1))$. For $k > \beta > 2$ we have

$$\frac{\beta k}{k+\beta-1} > \frac{\beta^2}{2\beta-1} \geq \sqrt{\beta}\ .$$

The case $\beta > k > 2$ is symmetrical. It follows that the cost is at least $\min\{\ln(\beta), \ln(k)\}/2$, as desired. The lower bound follows by observing that the hit cost is the same in every round. □

From V-Paging to GRBM

Observe that V-Paging roughly describes a single phase of our algorithm. Our simplifications include (i) that the number of new colors is fixed from the start, (ii) that the volume gap only increases for one color at a time, (iii) that we have no block requests, and (iv) that we assume an oblivious adversary.

It follows from the work of Blum et al. that simplifications (i), (ii) and (iii) can be overcome with few modifications to their strategy. The challenge of dealing with a non-oblivious adversary is, however, a significant one. To see this, recall that an adaptive offline adversary is so strong that randomization does not help against it (see Section 1.2.1).

Our Follow-the-leader algorithm of Section 3.4.3 solve this challenge by considering block requests instead of volume cost. The sequence of block requests is defined prior to the random choices, hence one can apply a standard analysis outside of a few special events such as marking new colors. As the number of block requests to a color is proportional to the increase in volume cost up to a factor of $(1+\gamma)$(see Lemma 3.31), an upper bound on the volume cost follows. This implies the upper bound on the volume gap.

One idea for improving the competitive ratio is to apply algorithm RWM using the number of requests as reference point. We show that this does not give a satisfactory algorithm for any choice of $\beta > 1$. Consider a scenario with $n = 1$. Suppose the request sequence gives the first color a minimum number of requests. The other $k-1$ colors have seen (up to an additive 1) $\log\log k$ requests more than color 1 for most of the

request sequence. This means that the volume cost of any color 1 increases a factor of $(1+\gamma)^{\log\log k} \approx \log k$ slower than the volume cost of any other color. In order to keep a small expected volume cost, one could require that β is chosen in a way that

$$\nabla \mathrm{smin}_{\beta,1}(x)_1 \gg \frac{1}{2} \ .$$

This, however, forces that

$$(k-1)\beta^{-\ln\ln k} + 1 \ll 2 \ ,$$

which implies $\ln(\beta) \geq \frac{\ln(k-1)}{\ln\ln k}$. With Lemma 3.41, this means that the hit cost of the algorithm degrades to $\frac{\log k}{\log\log k}$.

It follows that, in order to improve the competitive ratio, we cannot simply observe the block requests. We must also react to the increase in volume cost (or volume gap) directly. This will likely require a more precise model of the adversary. One could, for instance, exploit the fact that, given an online can compute at any time the current increase in volume cost *for any possible set of random choices made in the past*. We believe that this might actually be necessary for defining a suitable probability distribution on the stale colors.

In addition to this, an improved algorithm for scheduling the servers is likely necessary. The V-Paging problem and its solution via the RWM algorithm suggest that one should look into other online learning algorithms. The connection of online learning to the paging problem has long been known, yet it recently gained attention, e.g., in context of the k-Server algorithm by Bubeck et al. [BCL$^+$18]. We suggest, in particular, to consider the elegant *Follow-the-Perturbed-Leader* algorithm by Kalai and Vempala [KV05]. This algorithm obtains similar guarantees to RWM, yet it only requires few random choices, which could simplify an adaptation to our problem.

4. Conclusions and Open Problems

In this thesis we analyzed scheduling in the context of parallel systems and scheduling in scenarios with a reordering buffer. We have developed new algorithms for solving problems arising in these scenarios and we analyzed their properties mathematically. The results we obtained are first and foremost theoretical in nature, yet we are confident that an improved understanding of the mathematical foundations advances practical implementations in like manner.

In Chapter 2 we analyzed scheduling in parallel systems through the lens of cut problems in graphs and hypergraphs. We studied Minimum Hypergraph Bisection, a fundamental problem in this area of research. We showed that MHB admits an $\tilde{\mathcal{O}}(\sqrt{n})$-approximation algorithm and we were able to strengthen this result if the hypergraph in question is uniform. Next, we demonstrated how to construct cut trees to approximate the entire graph. These trees can serve as a building block in algorithms for more complicated cut problems such as partitioning a graph into multiple parts of equal size. Our tree approximates vertex cuts in graphs with quality $\tilde{\mathcal{O}}(\sqrt{W})$, where W is the sum of (positive) vertex weights in the graph.

We complemented our algorithmic results with strong lower bounds for the approximability of balanced cut problems. Our reduction from Densest k-Subgraph to MHB showed that MHB is part of a class of problems that are believed to admit no approximation algorithms of polylogarithmic quality. Formally, we proved that the *Exponential Time Hypothesis* implies that no $n^{1/(\log\log n)^c}$-approximation algorithm exists for MHB, where c is a universal constant. Via a different reduction, we were able to show that the *Hypergraph Dense versus Random Hypothesis* implies that no $n^{1/4-\varepsilon}$-approximation algorithm for MHB exists. Finally, we gave unconditional lower bounds on the quality of cut trees. Most importantly, we showed that any cut tree for unweighted vertex cuts must have quality $\Omega\big(n^{1/3}\big)$. This establishes a strong contrast to the case of edge cuts, for which trees of polylogarithmic quality exist, see, e.g., Räcke and Shah [RS14].

In Chapter 3 we turned to the problem of scheduling tasks with a reordering buffer in the form of Generalized Reordering Buffer Management. It had previously been known that any online algorithm for this problem must have competitive ratio $\Omega(\log k +$

$\log\log b$). We developed a new algorithm whose competitive ratio of $\mathcal{O}(\log k(\log k + \log\log b))$ nearly reaches that lower bound. The algorithm is asymptotically optimal for constant k and its competitive ratio is a doubly exponential improvement over previous work.

On the technical side, we introduced a new linear programming formulation for GRBM. A new linear program is the cornerstone of our algorithm and it allowed us to split the scheduling problem at hand into two mostly independent parts. The base part was essentially Block-Devices problem. We combined this with a new procedure vaguely inspired by algorithms for the Paging problem such as *Marking*. We also discussed possible improvements of our algorithm that could lower its competitive ratio to an asymptotically optimal $\mathcal{O}(\log k + \log\log b)$, yet it remains unclear, whether or not these ideas can be implemented successfully.

We close the thesis by highlighting more concrete open problems arising in the context of our work.

4.1. Open Problems

Our hardness results for MHB suggest that strong approximation algorithms are unlikely to exist for the problem. Due to the wide range of applications of hypergraph cuts, studying whether better algorithms for restricted hypergraph classes exist seems to be an interesting and relevant direction for future research. Notably, it would be interesting to understand if real-world instances exhibit similar characteristics as the worst-case instances we identified. Realistic hypergraph partitioning instances could, e.g., be collected from numerical simulations in parallel high-performance computing.

On a more theoretical side, it would be interesting to find out if the balanced cuts become easier to approximate if the underlying hypergraph is sufficiently sparse. As a concrete example, consider a hypergraph that has $n - 1$ simple edges (forming a tree) and n hyperedges of size $\sqrt{n}$. Finding an unbalanced $\sqrt{n}$-cut in this graph without losing a factor of $\Omega(\sqrt{n})$ seems impossible to do with the techniques used in this work. One possible avenue for the search of a better algorithm is the log-density framework used by Chlamtáč et al. for Minimum k-Union and related problems [CDM17].

The search for improved algorithms for MHB can be complemented by furthering the development of stronger lower bounds on the approximability of the problem. It would be interesting to see if there exists a possibility to transform an inapproximability result for MHB into a similar result for Minimum Bisection. It is unknown whether Minimum Bisection admits a PTAS if the ETH holds, yet we showed in this thesis that the ETH

rules out the existence of $n^{1/(\log\log n)^c}$-approximation algorithms for MHB. In an arguably quite ambitious project, one could consider the hardness of bisection problems under the classical P $\neq$ NP conjecture. Ruling out a PTAS for Minimum Bisection of MHB under this conjecture would come as a breakthrough in our understanding of cut problems.

In the context of GRBM it would be interesting to extend our techniques to star metrics. In this scenario, the cost of a server movement may depend on the new color of the server. A $\mathcal{O}\big((\log\log b\gamma)^2\big)$-competitive algorithm for RBM on weighted stars is known, where γ is the ratio between smallest and largest edge [AIMR15]. Adapting the linear program, even for $k = 1$, is not straightforward, as one can no longer simulate a block operation at constant cost via two server movements. We suggest the following modifications to the linear program instead: Split a block operation to color c into a movement to the center of the star (variables $f_{c'}(t)$) and a movement from the center to color c (variable $y_c(t)$ as before). Formally, one could consider the constraints

$$\begin{aligned} y_c(t) &\leq \sum_{c'} f_{c'}(t) && \forall c, t \ , \\ f_c(t) &\leq \delta_c(t) && \forall c, t \ . \end{aligned}$$

With $y_c(t) \leq \sum_{c'} f_{c'}(t)$, a block operation can only be performed if enough server mass exists at the center. With $f_c(t) \leq \delta_c(t)$, the amount of mass traveling to the center from c is limited. A major challenge then lies in adapting the construction of the lower bound to the new linear program.

Recent results for Paging and related problems have shown improved online algorithms through the use of projection-based formulations, see, e.g., Bubeck et al. [BCL$^+$18]. It is unclear whether or not these techniques can lead to improved online algorithms for buffering problems. In particular, it would be fascinating to understand if one can achieve a polylogarithmic competitive ratio for RBM that only depends on the buffer size and not on the properties of the underlying metric space.

A. Appendix

A.1. Proofs of Claim 2.13

We use the standard Chernoff bounds for $X \sim \mathrm{Bin}(n,p)$ with $\mu = n \cdot p$:

$$\Pr[X \leq (1-\delta)\mu] \leq \exp(-\delta^2\mu/2) \qquad 0 \leq \delta \leq 1 \ , \tag{A.1}$$

$$\Pr[X \geq (1+\delta)\mu] \leq \exp(-\delta\mu/3) \qquad 1 \leq \delta \ . \tag{A.2}$$

Proof of Claim 2.13 (1). With high probablity $\mathcal{G}_{n,p,r}$ with $p = n^{1+\alpha-r}$ has vertex degree $\Theta(n^\alpha)$.

Proof. For a fixed vertex v there are $q := \binom{n-1}{r-1}$ edges that contain v. Observe that $(2r)^{-r}n^{r-1} \leq q \leq n^{r-1}$. Let X be the random variable counting the degree of X, i.e., $X \sim \mathrm{Bin}(q,p)$ and $\mu = p \cdot q \in [(2r)^{-r}n^\alpha, n^\alpha]$. Recall that we assumed r to be a constant for any of our random hypergraphs.

For a given $\beta \geq 2$, choose n large enough so that $\frac{1}{3}\big(1-\frac{1}{\beta}\big) \geq \log n/\mu$, then we apply Chernoff bound (A.2) which gives

$$\Pr[X \geq \beta n^\alpha] \leq \Pr[X \geq \beta\mu] \leq \exp\left(-\frac{(\beta-1)\mu}{3}\right) \leq \exp(-\beta \log n) = n^{-\beta} \ .$$

Similarly, for a given $\beta \geq 2$, choose n large enough so that $\frac{1}{2\beta}\big(1-\frac{1}{\beta}\big)^2 \geq \log n/\mu$, then we apply Chernoff bound (A.1) which gives

$$\begin{aligned}\Pr\big[X \leq (2r)^{-r}\tfrac{1}{\beta}n^\alpha\big] &\leq \Pr\big[X \leq \tfrac{1}{\beta}\mu\big] \\ &\leq \exp\left(-\left(1-\frac{1}{\beta}\right)^2\frac{\mu}{2}\right) \leq \exp(-\beta\log n) = n^{-\beta} \ . \qquad \square\end{aligned}$$

Proof of Claim 2.13 (2). Any set of n edges in a $\mathcal{G}_{n,p,r}$ with $p = n^{1+\alpha-r}$ covers at least $\Omega\big((n/p)^{1/r}\big)$ vertices, with high probability.

Proof. For a fixed set of k vertices, there are $q = \binom{k}{r} \leq k^r$ edges contained in their subhypergraph. At least n of them exist with probability at most

$$\binom{q}{n}p^n \leq \left(\frac{eqp}{n}\right)^n \leq \left(\frac{ek^r p}{n}\right)^n \quad .$$

Setting $k = \frac{1}{2e}(n/p)^{1/r}$, this probability is at most

$$\left(\frac{ek^r p}{n}\right)^n \leq \left(\frac{enp}{2^r e^r pn}\right)^n \leq 2^{-n} \quad .$$

A union bound implies that the probability that *any* set of k vertices contains at least n edges is at most

$$\binom{n}{k}2^{-n} \leq n^k 2^{-n} = 2^{k\log n - n} \quad .$$

As $k \leq (n/p)^{1/r}$ and $p = n^{1+\alpha-r}$, we have $k \leq n^{1-\alpha/r}$, therefore the above probability is at most 2^{-cn} for some small c. This implies that the claim holds with high probability. □

Proof of Claim 2.13 (3). Any set of $n^{(1+\alpha)/2}/r$ edges in a $\mathcal{G}_{n,p,r}$ with $p = n^{1+\alpha-r}$ covers at least $\Omega\big(n^{(1+\alpha)/2-\varepsilon}\big)$ vertices, with high probability, if $\alpha < 1$, r is sufficiently large and ε is a small constant.

Proof. Any fixed set of k vertices has $q = \binom{k}{r} \leq k^r$ edges in its subhypergraph. At least $z = n^{(1+\alpha)/2}/r$ of them exist with probability at most

$$\binom{q}{z}p^z \leq \left(\frac{eqp}{z}\right)^z = \left(\frac{rek^r n^{1+\alpha-r}}{n^{(1+\alpha)/2}}\right)^{n^{(1+\alpha)/2}} \quad .$$

With $k = n^{(1+\alpha)/2-\varepsilon}/e$, this is

$$\left(\frac{ren^{r(1+\alpha)/2-r\varepsilon}n^{1+\alpha-r}}{e^r n^{(1+\alpha)/2}}\right)^{n^{(1+\alpha)/2}} \leq \left(n^{(r+1)(1+\alpha)/2-r-r\varepsilon}\right)^{n^{(1+\alpha)/2}}$$

We have used the fact that $re/e^r < 1$ for $r \geq 2$. A union bound implies that the probability that *any* set of k vertices contains at least $n^{(1+\alpha)/2}$ edges is at most

$$\binom{n}{k}\left(n^{(r+1)(1+\alpha)/2-r-r\varepsilon}\right)^{n^{(1+\alpha)/2}}$$
$$\leq n^{n^{(1+\alpha)/2}}\left(n^{(r+1)(1+\alpha)/2-r-r\varepsilon}\right)^{n^{(1+\alpha)/2}} \quad .$$

Observe that $(r+1)(1+\alpha)/2 - r + 1 \leq 0$ iff $r > (\alpha+3)/(1-\alpha)$. So, for r large enough, the above probability is at most $n^{-\varepsilon n^{(1+\alpha)/2}}$. This implies that the claim holds with high probability. □

A.2. Proofs of Claim 3.39

For $x \in \mathbb{R}^k$, positive integer $n \leq k$, and $\beta > 1$, define the β-softmin of order n as

$$\operatorname{smin}_{\beta,n}(x) = -\log_\beta\left(\sum_{T\in\binom{[k]}{n}} \beta^{-\sum_{i\in T} x_i}\right) ,$$

where $\binom{S}{n}$ denotes the set of n-tuples from the set S, and $[k] = \{1,\ldots,k\}$.

We will use that the gradient $\nabla\operatorname{smin}_{\beta,n}(x)$ of the β-softmin is

$$\nabla\operatorname{smin}_{\beta,n}(x)_j = \beta^{-x_j}\frac{\sum_{T\in\binom{[k]\setminus\{i\}}{n-1}} \beta^{-\sum_{i\in T} x_i}}{\sum_{T\in\binom{[k]}{n}} \beta^{-\sum_{i\in T} x_i}} \tag{A.3}$$

Proof of Claim 3.39(a). The function approximates the sum of the m smallest entries in x. Formally for all vectors x we have

$$\min_{T\in\binom{[k]}{n}} \textstyle\sum_{i\in T} x_i - n\log_\beta k \leq \operatorname{smin}_{\beta,n}(x) \leq \min_{T\in\binom{[k]}{n}} \sum_{i\in T} x_i \ .$$

Proof. To see the upper bound, observe that for any n-tuple T^*,

$$\begin{aligned}\operatorname{smin}_{\beta,n}(x) &= -\log_\beta\left(\sum_{T\in\binom{[k]}{n}} \beta^{-\sum_{i\in T} x_i}\right)\\ &\leq -\log_\beta\left(\beta^{-\sum_{i\in T^*} x_i}\right) = \min_{T\in\binom{[k]}{n}} \sum_{i\in T} x_i \ .\end{aligned}$$

For the upper bound let $y = \min_{T\in\binom{[k]}{n}} \sum_{i\in T} x_i$, then

$$\operatorname{smin}_{\beta,n}(x) \geq -\log_\beta\left(\binom{k}{n}\beta^{-y}\right) \geq y - n\log_\beta k \ .$$ □

Proof of Claim 3.39(b). The gradient of the function is the type of measure we require. Formally, for all vectors x, $0 \leq \nabla\operatorname{smin}_{\beta,n}(x)_c \leq 1$, and $\sum_c \nabla\operatorname{smin}_c(x)_c = n$.

Proof. Both properties follow immediately from Equation A.3 □

Proof of Claim 3.39(c). The change of the function is reasonably well approximated by its gradient. Formally for all vectors x, ℓ, if $\|\ell\|_1 \leq 1$ then

$$\mathrm{smin}_{\beta,n}(x+\ell) - \mathrm{smin}_{\beta,n}(x) \geq \frac{1-1/\beta}{\ln(\beta)} \nabla \mathrm{smin}_{\beta,n}(x)^T \ell \ .$$

Proof. We first show the lower bound.

Define $B_T(x) = \beta^{-\sum_{i\in T} x_i}$ for any vector x and $T \in \binom{[k]}{n}$ and $A(x) = \sum_T \beta^{-\sum_{i\in T} x_i}$. As $B_T(x+\ell) = B_T(x) \cdot B_T(\ell)$, we obtain

$$\begin{aligned}
\mathrm{smin}_{\beta,n}(x+\ell) - \mathrm{smin}_{\beta,n}(x) &= -\frac{1}{\ln(\beta)} \ln\left(\frac{1}{A(x)} \sum_{T\in\binom{[k]}{n}} B_T(x+\ell)\right) \\
&= -\frac{1}{\ln(\beta)} \ln\left(\frac{1}{A(x)} \sum_{T\in\binom{[k]}{n}} \Big(B_T(x) + B_T(x)(B_T(\ell)-1)\Big)\right) \\
&= -\frac{1}{\ln(\beta)} \ln\left(1 + \frac{1}{A(x)} \sum_{T\in\binom{[k]}{n}} B_T(x)(B_T(\ell)-1)\right) \\
&\geq \frac{1}{A(x)\ln(\beta)} \sum_{T\in\binom{[k]}{n}} B_T(x)(1-B_T(\ell)) \ .
\end{aligned}$$

The last step follows by applying the standard inequality $\ln(1+z) \leq z$. Another standard inequality, $1-\beta^{-z} \geq (1-\beta^{-1})z$ for $0 \leq z \leq 1$, implies that $1 - B_T(\ell)$ is at least $(1-\beta^{-1})\sum_{i\in T} \ell_i$, when $\ell \geq 0$ and $\|\ell\|_1 \leq 1$. It follows that

$$\begin{aligned}
\mathrm{smin}_{\beta,n}(x+\ell) - \mathrm{smin}_{\beta,n}(x) &\geq \frac{1-1/\beta}{A(x)\ln(\beta)} \sum_{T\in\binom{[k]}{n}} \sum_{i\in T} \ell_i B_T(x) \\
&= \frac{1-1/\beta}{A(x)\ln(\beta)} \sum_{j\in[k]} \ell_j e^{-x_j} \sum_{T\in\binom{[k]\setminus\{j\}}{n-1}} B_T(x) \\
&= \frac{1-1/\beta}{\ln(\beta)} \nabla \mathrm{smin}_{\beta,n}(x)^T \ell \ ,
\end{aligned}$$

where we regroup the sum according to ℓ_j in the second step. □

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[ACER11] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. Almost tight bounds for reordering buffer management. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 607–616, 2011. doi:10.1145/1993636.1993717.

[ACER12] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. Optimal online buffer scheduling for block devices. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC)*, pages 589–598, 2012. doi:10.1145/2213977.2214031.

[ACMM05] Amit Agarwal, Moses Charikar, Konstantin Makarychev, and Yury Makarychev. O(sqrt(log n)) approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. In *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, pages 573–581, 2005. doi:10.1145/1060590.1060675.

[ACN00] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1-2):203–218, 2000. doi:10.1016/S0304-3975(98)00116-9.

[AEGK14] Yossi Azar, Matthias Englert, Iftah Gamzu, and Eytan Kidron. Generalized reordering buffer management. In *Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 87–98, 2014. doi:10.4230/LIPIcs.STACS.2014.87.

[AEvSV19] Yossi Azar, Yuval Emek, Rob van Stee, and Danny Vainstein. The price of clustering in bin-packing with applications to bin-packing with delays. In *Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–10, 2019. doi:10.1145/3323165.3323180.

[AGG+09] Konstantin Andreev, Charles Garrod, Daniel Golovin, Bruce M. Maggs, and Adam Meyerson. Simultaneous source location. *ACM Transactions on Algorithms*, 6(1):16:1–16:17, 2009. doi:10.1145/1644015.1644031.

[AGGP17] Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proceedings of the 49th ACM Symposium on Theory of Computing (STOC)*, pages 551–563, 2017. doi:10.1145/3055399.3055475.

[AIMR15] Noa Avigdor-Elgrabli, Sungjin Im, Benjamin Moseley, and Yuval Rabani. On the randomized competitive ratio of reordering buffer management with non-uniform costs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 78–90, 2015. doi:10.1007/978-3-662-47672-7_7.

[AR06] Konstantin Andreev and Harald Räcke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006. doi:10.1007/s00224-006-1350-7.

[AR13] Noa Avigdor-Elgrabli and Yuval Rabani. An optimal randomized online algorithm for reordering buffer management. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1–10, 2013. doi:10.1109/FOCS.2013.9.

[AR15] Noa Avigdor-Elgrabli and Yuval Rabani. An improved competitive algorithm for reordering buffer management. *ACM Transactions on Algorithms*, 11(4):35:1–35:15, 2015. doi:10.1145/2663347.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM*, 56(2):5:1–5:37, 2009. doi:10.1145/1502793.1502794.

[AT19] Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 60–71, 2019. doi:10.1109/FOCS.2019.00013.

[BB02] Daniel K. Blandford and Guy E. Blelloch. Index compression through document reordering. In *Proceedings DCC 2002. Data Compression Conference*, pages 342–351, 2002. doi:10.1109/DCC.2002.999972.

[BBB+16] Marcin Bieńkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Nguyen Kim

Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. In *Proceedings of the 24th European Symposium on Algorithms (ESA)*, pages 12:1–12:17, 2016. doi:10.4230/LIPIcs.ESA.2016.12.

[BBC+14] Marcin Bieńkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. Better approximation bounds for the joint replenishment problem. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 42–54, 2014. doi:10.1137/1.9781611973402.4.

[BBJ+18] Marcin Bieńkowski, Martin Böhm, Lukasz Jez, Pawel Laskos-Grabowski, Jan Marcinkowski, Jirí Sgall, Aleksandra Spyra, and Pavel Veselý. Logarithmic price of buffer downscaling on line metrics. *Theoretical Computer Science*, 707:89–93, 2018. doi:10.1016/j.tcs.2017.10.008.

[BBK+94] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994. doi:10.1007/BF01294260.

[BBK99] Avrim Blum, Carl Burch, and Adam T. Kalai. Finely-competitive paging. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 450–458, 1999. doi:10.1109/SFFCS.1999.814617.

[BBMN15] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *Journal of the ACM*, 62(5):40:1–40:49, 2015. doi:10.1145/2783434.

[BBN12] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *Journal of the ACM*, 59(4):19:1–19:24, 2012. doi:10.1145/2339123.2339126.

[BCC+10] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 201–210, 2010. doi:10.1145/1806689.1806719.

[BCL+18] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *Proceedings of the 50th ACM Symposium on Theory of Computing (STOC)*, pages 3–16, 2018. doi:10.1145/3188745.3188798.

[BCV+12] Aditya Bhaskara, Moses Charikar, Aravindan Vijayaraghavan, Venkatesan Guruswami, and Yuan Zhou. Polynomial integrality gaps for strong SDP relaxations of densest k-subgraph. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 388–405, 2012. doi:10.1137/1.9781611973099.34.

[BE98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[BFNT17] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1235–1244, 2017. doi:10.1137/1.9781611974782.80.

[BMS+16] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 117–158. Springer, 2016. doi:10.1007/978-3-319-49487-6_4.

[BN09] Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009. doi:10.1561/0400000024.

[CDK12] Eden Chlamtáč, Michael Dinitz, and Robert Krauthgamer. Everywhere-sparse spanners via dense subgraphs. In *Proceedings of the 53rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 758–767, 2012. doi:10.1109/FOCS.2012.61.

[CDM17] Eden Chlamtáč, Michael Dinitz, and Yury Makarychev. Minimizing the union: Tight approximations for small set bipartite vertex expansion. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 881–899, 2017. doi:10.1137/1.9781611974782.56.

[CFLP00] Robert D. Carr, Lisa Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 106–115, 2000. URL `http://dl.acm.org/citation.cfm?id=338219.338241`.

[CMSvS12] Ho-Leung Chan, Nicole Megow, René Sitters, and Rob van Stee. A note on sorting buffers offline. *Theoretical Computer Science*, 423:11–18, 2012. doi:10.1016/j.tcs.2011.12.077.

[Cod60] Edgar F. Codd. Multiprogram scheduling parts 1 and 2: Introduction and theory. *Communications of the ACM*, 3(6):347–350, 1960. doi:10.1145/367297.367317.

[CPSV18] Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. The online set aggregation problem. In *Proceedings of the 13st Latin American Symposium on Theoretical Informatics (LATIN)*, pages 245–259, 2018. doi:10.1007/978-3-319-77404-6_19.

[CX17] Chandra Chekuri and Chao Xu. Computing minimum cuts in hypergraphs. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1085–1100, 2017. doi:10.1137/1.9781611974782.70.

[Din16] Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:128, 2016. URL `http://eccc.hpi-web.de/report/2016/128`.

[EKW16] Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the 48th ACM Symposium on Theory of Computing (STOC)*, pages 333–344, 2016. doi:10.1145/2897518.2897557.

[EMPS16] Alina Ene, Gary L. Miller, Jakub Pachocki, and Aaron Sidford. Routing under balance. In *Proceedings of the 48th ACM Symposium on Theory of Computing (STOC)*, pages 598–611, 2016. doi:10.1145/2897518.2897654.

[Eng18] Matthias Englert. The reordering buffer problem on the line revisited. *SIGACT News*, 49(1):67–72, 2018. doi:10.1145/3197406.3197418.

[ER17] Matthias Englert and Harald Räcke. Reordering buffers with logarithmic diameter dependency for trees. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1224–1234, 2017. doi:10.1137/1.9781611974782.79.

[ERS19] Matthias Englert, Harald Räcke, and Richard Stotz. Polylogarithmic guarantees for generalized reordering buffer management. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 38–59, 2019. doi:10.1109/FOCS.2019.00012.

[ERW09] Matthias Englert, Heiko Röglin, and Matthias Westermann. Evaluation of online strategies for reordering buffers. *ACM Journal of Experimental Algorithmics*, 14, 2009. doi:10.1145/1498698.1564503.

[ERW10] Matthias Englert, Harald Räcke, and Matthias Westermann. Reordering buffers for general metric spaces. *Theory of Computing*, 6(1):27–46, 2010. doi:10.4086/toc.2010.v006a002.

[EW05] Matthias Englert and Matthias Westermann. Reordering buffer management for non-uniform cost models. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 627–638, 2005. doi:10.1007/11523468_51.

[Fei02] Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC)*, pages 534–543, 2002. doi:10.1145/509907.509985.

[FF15] Andreas Emil Feldmann and Luca Foschini. Balanced partitions of trees and applications. *Algorithmica*, 71(2):354–376, 2015. doi:10.1007/s00453-013-9802-3.

[FHL08] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008. doi:10.1137/05064299X.

[FK02] Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal on Computing*, 31(4):1090–1118, 2002. doi:10.1137/S0097539701387660.

[FKL+91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991. doi:10.1016/0196-6774(91)90041-V.

[FKN00] Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. Approximating the minimum bisection size (extended abstract). In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC)*, pages 530–536, 2000. doi:10.1145/335305.335370.

[FMP+04] Tomás Feder, Rajeev Motwani, Rina Panigrahy, Steven S. Seiden, Rob van Stee, and An Zhu. Combining request scheduling with web caching. *Theoretical Computer Science*, 324(2-3):201–218, 2004. doi:10.1016/j.tcs.2004.05.016.

[FRR94] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48(3):410–428, 1994. doi:10.1016/S0022-0000(05)80060-1.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004. doi:10.1016/j.jcss.2004.04.011.

[GH61] Ralph E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. doi:10.1137/0109047.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and intractability : A guide to the theory of NP-completeness*. W. H. Freeman, 1979.

[GJS76] Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

[GS09] Iftah Gamzu and Danny Segev. Improved online algorithms for the sorting buffer problem on line metrics. *ACM Transactions on Algorithms*, 6(1), 2009. doi:10.1145/1644015.1644030.

[HHR03] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 34–43, 2003. doi:10.1145/777412.777419.

[HK00] Bruce Hendrickson and Tamara G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26(12):1519–1534, 2000. doi:10.1016/S0167-8191(00)00048-X.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

[Kho06] Subhash Khot. Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM Journal on Computing*, 36(4):1025–1071, 2006. doi:10.1137/S0097539705447037.

[KNR02] Sanjeev Khanna, Joseph Naor, and Danny Raz. Control message aggregation in group communication protocols. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 135–146, 2002. doi:10.1007/3-540-45465-9_13.

[KNS09] Robert Krauthgamer, Joseph Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 942–949, 2009. doi:10.1137/1.9781611973068.102.

[Kou09] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009. doi:10.1016/j.cosrev.2009.04.002.

[KP95] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

[KPR93] Philip N. Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*, pages 682–690, 1993. doi:10.1145/167088.167261.

[KR17] Matthias Kohler and Harald Räcke. Reordering buffer management with a logarithmic guarantee in general metric spaces. In *Proceedings of the 44nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 33:1–33:12, 2017. doi:10.4230/LIPIcs.ICALP.2017.33.

[KRSW04] Jens Krokowski, Harald Räcke, Christian Sohler, and Matthias Westermann. Reducing state changes with a pipeline buffer. In *Proceedings of the 9th International Fall Workshop Vision, Modeling, and Visualization (VMV)*, pages 217–224, 2004.

[KV05] Adam Kalai and Santosh S. Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005. doi:10.1016/j.jcss.2004.10.016.

[Lee19] Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Mathematical Programming*, 177(1-2):1–19, 2019. doi:10.1007/s10107-018-1255-7.

[LR99] Frank T. Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999. doi:10.1145/331524.331526.

[Man17] Pasin Manurangsi. Almost-polynomial ratio ETH-hardness of approximating densest k-subgraph. In *Proceedings of the 49th ACM Symposium on Theory of Computing (STOC)*, pages 954–961, 2017. doi:10.1145/3055399.3055412.

[Man18] Pasin Manurangsi. Inapproximability of maximum biclique problems, minimum k-cut and densest at-least-k-subgraph from the small set expansion hypothesis. *Algorithms*, 11(1):10, 2018. doi:10.3390/a11010010.

[MMS90] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.

[MS90] David W. Matula and Farhad Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics*, 27(1-2):113–123, 1990. doi:10.1016/0166-218X(90)90133-W.

[MS91] Lyle A. McGeoch and Daniel D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991. doi:10.1007/BF01759073.

[Räc02] Harald Räcke. Minimizing congestion in general networks. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 43–52, 2002. doi:10.1109/SFCS.2002.1181881.

[Räc08] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC)*, pages 255–264, 2008. doi:10.1145/1374376.1374415.

[RS14] Harald Räcke and Chintan Shah. Improved guarantees for tree cut sparsifiers. In *Proceedings of the 22th European Symposium on Algorithms (ESA)*, pages 774–785, 2014. doi:10.1007/978-3-662-44777-2_64.

[RS16] Harald Räcke and Richard Stotz. Improved approximation algorithms for balanced partitioning problems. In *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 58:1–58:14, 2016. doi:10.4230/LIPIcs.STACS.2016.58.

[RSS18] Harald Räcke, Roy Schwartz, and Richard Stotz. Trees for vertex cuts, hypergraph cuts and minimum hypergraph bisection. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 23–32, 2018. doi:10.1145/3210377.3210398.

[RSW02] Harald Räcke, Christian Sohler, and Matthias Westermann. Online scheduling for sorting buffers. In *Proceedings of the 10th European Symposium on Algorithms (ESA)*, pages 820–832, 2002. doi:10.1007/3-540-45749-6_71.

[SGV04] Sven Spieckermann, Kai Gutenschwager, and Stefan Voß. A sequential ordering problem in automotive paint shops. *International Journal of Production Research*, 42(9):1865–1878, 2004. doi:10.1080/00207540310001646821.

[SS13] Peter Sanders and Christian Schulz. Think locally, act globally: Highly balanced graph partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA)*, pages 164–175, 2013. doi:10.1007/978-3-642-38527-8_16.

[ST85] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

[SV95] Huzur Saran and Vijay V. Vazirani. Finding k cuts within twice the optimal. *SIAM Journal on Computing*, 24(1):101–108, 1995. doi:10.1137/S0097539792251730.

[Vij12] Aravindan Vijayaraghavan. *Beyond Worst Case Analysis in Approximation Algorithms*. PhD thesis, Princeton University, 2012. URL `http://arks.princeton.edu/ark:/88435/dsp01qr46r086z`.

[Wil19] R. Ryan Williams. Some estimated likelihoods for computational complexity. In *Computing and Software Science - State of the Art and Perspectives*, volume 10000 of *Lecture Notes in Computer Science*, pages 9–26. Springer, 2019. doi:10.1007/978-3-319-91908-9_2.