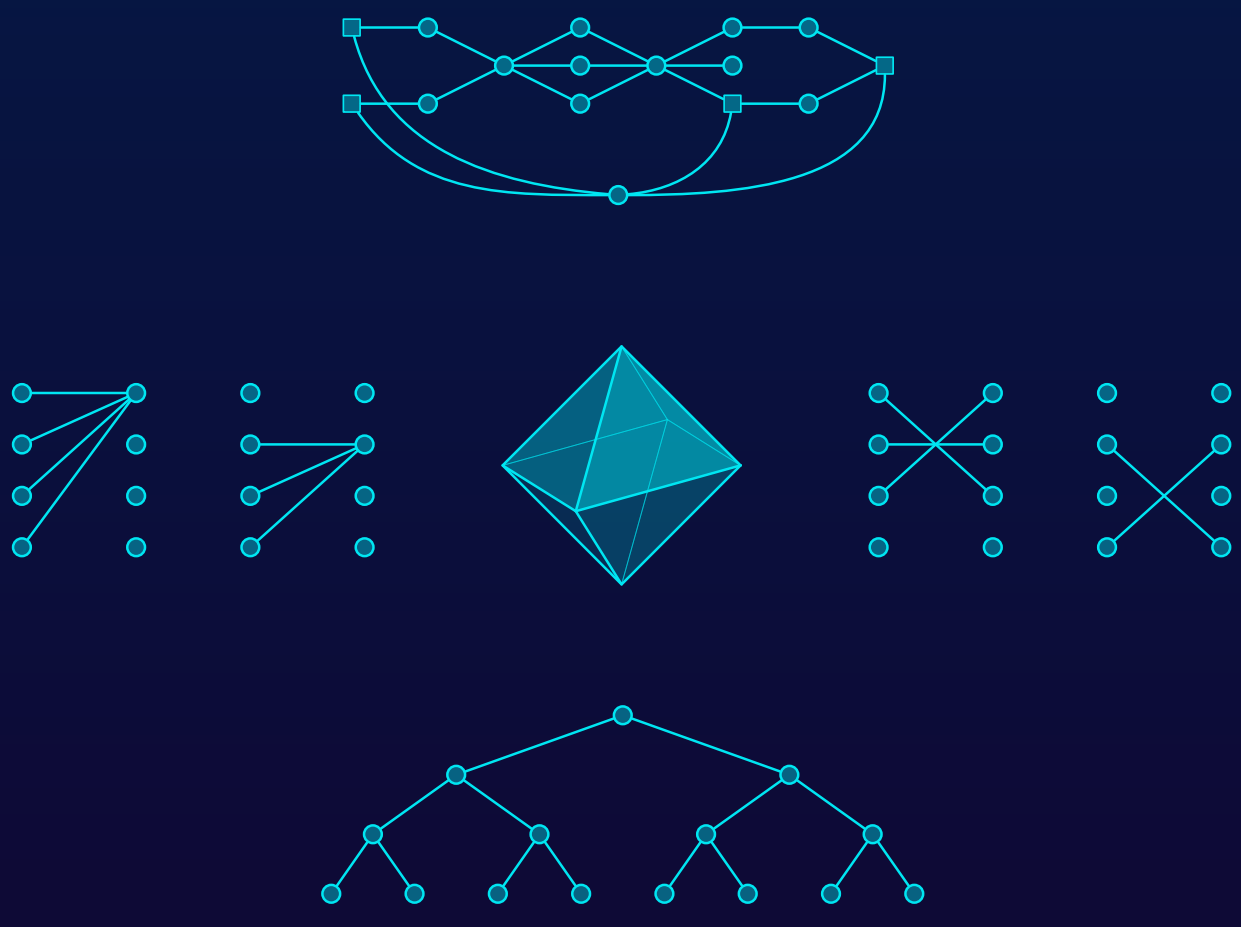
Approximation via Duality in Offline, Online and Strategic Settings



Danish Kashaev

Approximation via Duality in Offline, Online and Strategic Settings

Danish Kashaev

Approximation via Duality in Offline, Online and Strategic Settings

ILLC Dissertation Series DS-202X-NN

Institute for Logic, Language and Computation

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107
1098 XG Amsterdam
phone: +31-20-525 6051
e-mail: illc@uva.nl
homepage: http://www.illc.uva.nl/

The research for this doctoral thesis has been done in the Networks and Optimization research group at the Centrum Wiskunde & Informatica (CWI) in Amsterdam and received funding from the Netherlands Organization for Scientific Research (NWO) through the OPTIMAL (Optimization for and with Machine Learning) project.

Printed and bound by Ipskamp Printing

ISBN: 978-94-6536-004-1

Approximation via duality in offline, online and strategic settings

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op dinsdag 10 februari 2026, te 16.00 uur

door Danish Kashaev
geboren te Helsinki

Promotiecommissie

Promotores:	prof. dr. G. Schäfer	Universiteit van Amsterdam
	prof. dr. D.N. Dadush	Universiteit Utrecht
Overige leden:	prof. dr. K.R. Apt	Universiteit van Amsterdam
	dr. G. Regts	Universiteit van Amsterdam
	dr. rer. nat. R.E.M. Reiffenhäuser	Universiteit van Amsterdam
	prof. dr. ir. K.I. Aardal	TU Delft
	dr. N.K. Olver	LSE
	prof. dr. M. Uetz	University of Twente
	prof. dr. R.M. de Wolf	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Әниемә һәм әтиемә (to my mom and to my dad)

Contents

Acknowledgments

The end of this PhD degree marks the end of four years of a very interesting journey in Amsterdam. There are numerous people I would like to thank.

I would first like to thank my advisor Guido Schäfer. Thank you for hiring me and giving me the opportunity of doing my thesis in this great research institute. I am grateful for the freedom you gave me in exploring my personal research interests, encouraging me to start collaborations, the support in the tough moments, the nice research discussions and the fun moments. I would also like to thank my co-advisor Daniel Dadush. Thank you for giving me an opportunity to collaborate together on the heap selection project, the experience of organizing the group seminar and the research discussions about cut-matching games.

Thank you to Krzysztof Apt, Guus Regts, Rebecca Reiffenhäuser, Karen Aardal, Neil Olver, Marc Uetz and Ronald de Wolf for agreeing to be part of my doctorate committee and for taking the time to read my thesis.

I was lucky to have the opportunity to work with great people during this time, and I thank all my collaborators. I would in particular like to mention Sander and Cedric (Zhuan Khye). I really enjoyed working with you on our joint projects and I feel very fortunate that our paths crossed at CWI during my time as a PhD student.

Thank you to all the nice colleagues and friends that I had during this time at CWI. Thank you Sven for being my first office mate, our funny foreign language sessions and for taking me to the Ajax games at the famous Johan Cruijff Arena. Thanks to Luis Felipe for our sports discussions, table tennis matches and cool trips to different places in the Netherlands. Thanks Samarth for the bouldering sessions and the poker discussions. Thanks Sebastiaan for a cool concert. Sid, thank you for all the tips about the German language. Jens, Ferenc, Simon, Sander, Cedric, Ruben, Akshay, it was always a pleasure to hang out or have a drink with you. Thanks also to all the other nice colleagues in our group at CWI, including Alexander, Andries, Artem, Arthur, Arthur, Ben, Constantinos, Corbi, Hilde, Ilker, Joakim, Jonas, Leen, Lucas, Mehmet, Michelle, Monique, Solon, Sophie, Sophie, Twan, Willem, Yasamin.

Thanks to Guido for a very nice course on algorithmic game theory and to Monique and Sven for another very interesting one on semidefinite programming. I spent a lot of time doing the homework sets at the beginning of my PhD for

these two courses and became interested in trying to use what I have learned in my research thanks to the nice lectures that you all gave.

Thank you to my housemates Dyon and Henrik. It was a pleasure living with you both, I found it very cool to share roughly the same stage of the PhD journey during all this time with fellow PhD students. I also really appreciate the fact that we spoke Dutch so often at home, it allowed me to improve at it fast and made the experience of living in Amsterdam more fulfilling.

Other people from CWI that I would like to thank are Minnie for the always entertaining conversations and Bikkie for the regular fun discussions at the library while I was getting the morning coffee. Also thanks to the different colleagues and fellow PhD students from the other research groups, including Ake, Alexander, Lisa, Llorenc, Pardeep, Rik, René, Simona, Syver, Toby, Quinten, Xuemei. I always enjoyed hanging out and having a chat with all of you.

Thanks to all the cool people I met at the different tennis clubs in Amsterdam, including Buitenveldert, Chip & Charge and Tiebreakers. It was refreshing to hang out with many interesting people outside of the scientific environment, and it was a good way to immerse myself more into the Dutch culture. Thanks to my friends Niels and Andre, I will definitely miss these early Sunday morning sessions.

It was also very cool to meet fellow Tatar people from different cities in the Netherlands. Thanks in particular to Ayrat, Almaz and Alsina for the fun meet-ups.

Thank you to all my close friends from back home in Switzerland. I was returning home quite often during my doctoral studies and I am glad that I managed to stay in touch with many of you while living abroad for a few years.

Finally, I would like to thank my family: my mom Goljihan, my dad Rinat, my sister Kamila and my brother Toufan. You have always been my biggest support and this thesis would not have been possible without you all.

Danish Kashaev
November 2025, Amsterdam

Abstract

The main questions studied in this thesis can be broadly stated as follows: given an NP-hard combinatorial optimization problem and a suboptimal solution to this problem – obtained, for instance, by an efficient approximation algorithm – how close can this solution get to the optimal one? The quality of a solution is measured by the worst-case ratio, over all input instances, of the cost of the solution to that of the optimal one.

The aim of this thesis is to develop new techniques for proving tight bounds on this question for three fundamental classes of combinatorial optimization problems: covering, matching, and scheduling. We moreover study this question in three different, yet related, contexts. The suboptimal solutions considered may be obtained by a standard approximation algorithm, which has access to the entire input upfront. In this case, the ratio of interest is called the approximation ratio. A more restrictive computational model reveals the input only partially over time, requiring an online algorithm to make irrevocable decisions at each step. In this setting, the relevant measure is the competitive ratio. Finally, the solution may arise as a game-theoretic equilibrium (for instance, a Nash equilibrium), in which case the relevant measure is called the price of anarchy.

A unifying theme in all our results is the use of convex programming relaxations, such as linear programming (LP) and semidefinite programming (SDP). In particular, we frequently leverage the power of convex programming duality to construct carefully chosen dual solutions that guide our various analyses and help design our algorithms.

We first initiate a beyond the worst-case analysis of the classical vertex cover problem and its standard LP relaxation. This problem is efficiently solvable on bipartite graphs, and a 2-approximation algorithm can be obtained by rounding the LP on general graphs. We introduce new parameters and consider an algorithm which attains bounds that interpolate between these two extremes. For three-colorable graphs, our result gives an understanding of when the integrality gap of the LP decreases to one, depending on the graph structure.

Next, we study a generalization of the classic online bipartite matching problem to hypergraphs, focusing specifically on 3-uniform hypergraphs under online vertex arrivals. We present an optimal primal-dual fractional algorithm for this problem and complement it with the construction of an adversarial instance that

establishes a matching upper bound. We also provide a better than greedy randomized integral algorithm when the online nodes have bounded degree.

We then consider several scheduling and congestion problems under the objective of minimizing the sum of weighted completion times. We introduce a dual fitting framework on a single semidefinite program which simultaneously yields simple proofs of tight bounds for the approximation ratio of local search algorithms, the competitive ratio of online algorithms, and the price of anarchy of games. Our results simplify and unify important known results through this unified framework.

Finally, we consider an online graph exploration problem on a binary heap related to the branch and bound algorithm to solve integer programs. We provide a new randomized algorithm improving the best known running time for this problem at the expense of slightly increased space usage.

Chapter 1

Introduction

A mathematical optimization problem involves finding a solution that minimizes or maximizes an objective function, subject to a set of constraints. Such problems lie at the core of both computer science and mathematics, and they appear in a wide variety of applications: from logistics and operations research to machine learning and economics. Over the years, a rich arsenal of algorithmic techniques has been developed to address these problems. For many optimization problems, researchers have discovered *efficient* algorithms, meaning algorithms that run in time polynomial in the size of the input.

However, many fundamental optimization problems have been shown to be NP-hard, meaning that they are unlikely to admit efficient algorithms to compute an optimal solution for every instance. As a consequence, the field of *approximation algorithms* has emerged as a central area of research, aiming to design efficient algorithms that compute solutions whose quality can be proven to be close to optimal.

Approximation algorithms are important for numerous reasons. In practice, many optimization problems are NP-hard, and exact optimal solutions are computationally out of reach for large instances. Having an approximation guarantee provides a *worst-case* bound on how far a solution can be from optimal. This is crucial in real-world applications where solutions must meet performance thresholds, even under uncertainty or adversarial conditions. Approximation guarantees can also act as an objective benchmark for algorithms. If an algorithm achieves a certain approximation ratio, it becomes a standard against which others are measured. This drives progress in algorithm design and helps classify problems by their approximability. This, in turn, allows to build a theory understanding which algorithmic problems are fundamentally more difficult than others.

In addition to that, approximation algorithms are very interesting from a mathematical point of view. The field is rich with deep connections having been found to other areas of mathematics such as combinatorics, graph theory, convex optimization, probability or game theory. Over the past decades, this area has seen significant developments: a wide array of approximation techniques such as greedy methods, dynamic programming, linear and semidefinite programming

relaxations, local search, and primal-dual frameworks have been developed which led to algorithms with tight guarantees for numerous classical problems.

While worst-case approximation guarantees provide a foundational understanding of algorithmic performance, they can sometimes be too pessimistic. It can happen that algorithms with poor worst-case guarantees perform remarkably well in practice, on real-world or structured inputs. This motivates the study of beyond worst-case models, which seek to explain an algorithm's performance under different assumptions. These models include smoothed analysis, parameterized complexity, stochastic inputs, or inputs augmented with machine learned predictions. Beyond worst-case analysis provides more nuanced performance guarantees, and can often lead to new insights, as well as to the development of new algorithmic techniques.

In many real-world applications, decisions must be made sequentially, without knowledge of the entire input upfront. This is the setting captured by *online algorithms*, where the input arrives sequentially over time, and an online algorithm needs to make irrevocable choices based only on the information available so far. Such models are central to a wide range of practical problems, including load balancing in cloud computing, caching in memory systems, ride-sharing or online auctions. The challenge lies in performing nearly as well as an optimal offline algorithm that has full knowledge of the entire input sequence in advance. The performance of an online algorithm is compared to the optimal offline benchmark, and the measure of quality is called the *competitive ratio*. It is defined as the worst-case, over all online inputs, of the ratio between the cost of an online algorithm and the cost of an optimal offline solution.

Some algorithmic settings also involve strategic agents aiming to minimize their own cost, often at the expense of global efficiency. Real world examples include network routing, cloud resource allocation, auctions or ad markets. In such systems, it becomes interesting to study equilibrium outcomes, where no agent has an incentive to deviate from their chosen strategy. In this setting, the *price of anarchy* provides a formal measure of how bad such equilibrium outcomes can be compared to the global optimal solution, also called the *social optimum*. It is formally defined as the ratio between the cost of a worst-case equilibrium and the cost of an optimal solution. This setting has connections to classical approximation algorithms theory, for instance when the output of an algorithm is guaranteed to be an equilibrium.

A lot of very interesting research has developed in the past decades on approximation algorithms and the mentioned related settings. There are however still many interesting open questions and possible research directions. The landscape of approximability is still only partially understood, particularly in models that go beyond the classical offline setting. Problems involving beyond worst-case models, online decisions or strategic agents introduce new challenges, for which new tools and modern techniques are still to be discovered.

The aim of this thesis is to contribute to the theory of approximation algo-

rithms by developing new techniques and results for different optimization problems, both in classical offline settings, as well as online or game-theoretic models. More specifically, we address problems in covering, matching, and scheduling, exploring both worst-case and beyond worst-case guarantees.

A unifying theme in our results is the use of *convex relaxations*, in particular of *linear programming* and *semidefinite programming*. These relaxations allow to find approximate fractional solutions to computationally hard problems in polynomial time. Beyond their algorithmic tractability, convex relaxations admit a rich duality theory, which provides deep insights into the structure of optimal solutions. Throughout this thesis, we frequently leverage convex programming duality as an analytical tool, as well as a guiding principle in the design and analysis of our algorithms.

Overview of the results

Beyond the worst-case approximation for vertex cover

The weighted vertex cover problem is one of the most fundamental NP-hard combinatorial optimization problems. A classical result shows that it admits a simple 2-approximation by rounding an optimal solution to the standard linear programming relaxation. On the hardness side, the problem is known to be $2-\varepsilon$ hard to approximate, assuming the well known unique games conjecture. In the special case where the input graph is bipartite, the LP relaxation becomes tight: the underlying polytope is integral, implying that the problem can be solved exactly in polynomial-time on bipartite graphs.

We initiate a beyond the worst-case study of the problem by assuming an oracle access to an *induced bipartite subgraph* of the input graph. Equivalently, we assume that we have knowledge of a subset of vertices whose removal leaves the remaining graph bipartite. We tightly analyze a natural algorithm which takes this subset of vertices to the solution, and then solves the problem on the remaining bipartite graph optimally. The approximation guarantee obtained is a bound "interpolating" between the worst-case bound of 2 and the optimal bound of 1 for bipartite graphs, and takes the form $(1+1/\rho)(1-\alpha)+2\alpha$, where $\rho \in [2,\infty]$ and $\alpha \in [0,1]$. The parameter ρ is related to the odd girth of the graph and measures how far the input graph is from being bipartite, whereas the parameter α is a measure of the "quality" of the bipartizing set. For three-colorable graphs, we have $\alpha = 0$ and the bound becomes $1 + 1/\rho$, offering an understanding of how the integrality gap of the standard LP decreases depending on the input graph structure. We believe the most interesting part of this work are the techniques used to prove this result, by leveraging LP duality to optimize over the space of feasible weight functions, and hope that similar techniques can be applied to other problems.

Online matching on hypergraphs

A more restrictive computational model, known as the *online* model, reveals the input to an optimization problem incrementally over time. In this setting, an online algorithm must make irrevocable decisions at each step without knowledge of future inputs. A central topic in this field is online matching, first introduced in [KVV90]. In the classical version, one side of a bipartite graph is known in advance, while vertices on the other side arrive one by one. Upon each arrival, the algorithm must decide immediately and irrevocably whether to match the arriving vertex to an available neighbor. The goal is to produce a matching, i.e. a disjoint subset of edges, of maximum cardinality. It is known that the right competitive ratio is $1 - 1/e$ for both integral and fractional versions of this problem.

Since then, a rich body of work has developed around online graph matching and its variants. However, far less is known about online matching in hypergraphs, where the complexity of decision-making increases significantly. We study a three-dimensional generalization of the original problem introduced for bipartite graphs in the vertex arrival model. Our main contribution is to provide a $(e-1)/(e+1)$-competitive fractional algorithm, along with a matching upper bound based on a carefully constructed adversarial instance. The integral case remains a very intriguing open problem, where the best known algorithm to date is still the simple greedy strategy. However, we provide improved guarantees for the integral case under the assumption that arriving vertices have bounded degree.

The main technical contribution of this work is the construction of the adversarial instance, which combines two hard instances for bipartite graphs under the vertex arrival and edge arrival models. We moreover develop a more fine-grained understanding of the behaviour of an arbitrary fractional algorithm, needed to address the increased complexity of the hypergraph setting.

Semidefinite dual fitting for scheduling problems

Scheduling is a fundamental problem in computer science and operations research, with applications ranging from cloud computing and manufacturing to traffic control and network routing. The problem consists of assigning and scheduling a set of jobs to a set of machines in order to optimize some objective function. In many real-world scenarios, decisions must be made without complete knowledge of future events or with the participation of self-interested agents. We investigate several scheduling problems in two such settings: the online setting, where jobs arrive over time and decisions must be made irrevocably, and the game-theoretic setting, where jobs are controlled by selfish agents aiming to minimize their own cost.

Our main contribution is a unified dual fitting framework based on a single semidefinite program (SDP), which we use to derive tight bounds in both the

online and game-theoretic settings for scheduling problems. This same approach also proves useful in the classical offline setting, where it allows to tightly analyze local search algorithms, a powerful class of combinatorial algorithms. Using this structure, we are able to give simple and unified proofs of numerous important results in the field, which include the analysis of the price of anarchy of the unrelated machine scheduling problem $R||\sum w_j C_j$, the best known deterministic and randomized coordination mechanisms for this problem, as well the best known combinatorial approximation algorithm in the offline setting. In the game theoretic setting, we also recover the price of anarchy of weighted affine congestion games and the pure price of anarchy of scheduling on parallel machines. In the online setting, we recover the best known deterministic and randomized algorithms for the online load balancing problem on unrelated machines, provide an improved fractional algorithm, and complement this with matching lower bounds. We also introduce a new model, for which we present an optimal algorithm.

A faster algorithm for explorable heap selection

We study an online graph exploration problem on a heap related to the branch and bound algorithm. The input is a binary heap, with key values associated to every node. The key values can only be accessed by traversing through the underlying infinite binary tree, and the complexity of the algorithm is measured by the total distance traveled in the tree, where each edge has unit cost. This problem was originally proposed as a model to study search strategies for the branch-and-bound algorithm with storage restrictions by Karp, Saks and Widgerson [KSW86], who gave deterministic and randomized $n \cdot \exp(O(\sqrt{\log n}))$ time algorithms using $O(\log(n)^{2.5})$ and $O(\sqrt{\log n})$ space respectively. We present a new randomized algorithm with running time $O(n \log(n)^3)$ against an oblivious adversary using $O(\log n)$ space, substantially improving the previous best randomized running time at the expense of slightly increased space usage.

Thesis organization

The thesis is organized as follows. Chapter 2 is devoted to preliminaries needed to understand the main results. In Chapter 3, we study the vertex cover problem in a beyond the worst-case setting. Chapter 4 concerns the online matching problem on 3-uniform hypergraphs. In Chapter 5, we introduce the semidefinite programming dual fitting technique and show how it can be used to bound the price of anarchy of games and the approximation ratio of local search algorithms for scheduling problems. Chapter 6 extends this technique to help design and analyze online algorithms. Chapter 7 presents the faster randomized algorithm for the explorable heap selection problem.

Publications

Chapter 3 is based on joint work with Guido Schäfer and has appeared in [KS23]. Chapter 4 is based on joint work with Sander Borst and Zhuan Khye Koh and has appeared in [BKK25]. Chapter 5 is based on a work appearing in [Kas25]. Chapter 6 was at the time of writing still ongoing work, joint with Sander Borst, a part of which is included in [BK25]. Chapter 7 is based on joint work with Sander Borst, Daniel Dadush, Sophie Huiberts and has appeared in [BDHK23, BDHK24].

[KS23] Danish Kashaev and Guido Schäfer. Round and bipartize for vertex cover approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, volume 275. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023

[BKK25] Sander Borst, Danish Kashaev, and Zhuan Khye Koh. Online matching on 3-uniform hypergraphs. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 100–113. Springer, 2025

[Kas25] Danish Kashaev. Selfish, local and online scheduling via vector fitting. To appear in *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2026)*, 2026

[BDHK23] Sander Borst, Daniel Dadush, Sophie Huiberts, and Danish Kashaev. A nearly optimal randomized algorithm for explorable heap selection. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 29–43. Springer, 2023

[BDHK24] Sander Borst, Daniel Dadush, Sophie Huiberts, and Danish Kashaev. A nearly optimal randomized algorithm for explorable heap selection. *Mathematical Programming*, pages 1–22, 2024

[BK25] Sander Borst and Danish Kashaev. Improved online load balancing in the two-norm. *arXiv preprint arXiv:2511.03345*, 2025

Chapter 2

Preliminaries

2.1 Combinatorial optimization problems

A combinatorial (or discrete) optimization problem Π is either a minimization or a maximization problem. Each instance $\mathcal{I}$ of Π can often be described by a finite set N, called the *ground set*, a family of subsets of $\mathcal{F} \subseteq 2^N$, called the *feasible solutions*, and an objective function $w : \mathcal{F} \to \mathbb{Q}$ to minimize or maximize. Here are a few examples of fundamental combinatorial optimization problems.

Minimum weight spanning tree. The input is a connected graph $G = (V, E)$ along with a weight function $w : E \to \mathbb{R}_{\geq 0}$. The goal is to find a spanning tree $T \subseteq E$ of the graph, i.e. an acyclic connected subgraph covering all the nodes, of minimum weight $w(T) := \sum_{e \in T} w_e$.

Shortest path. The input is a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}_{\geq 0}$ and two distinguished nodes $s, t \in V$. The goal is to find a path $P \subseteq E$ between s and t in the graph of minimum weight $w(P) := \sum_{e \in P} w_e$.

Maximum weight matching. The input is a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}_{\geq 0}$. The goal is to find a disjoint subset of edges $M \subseteq E$ of maximum weight $w(M) := \sum_{e \in M} w_e$.

2.1.1 Algorithms and efficiency

An algorithm for an optimization problem Π is a sequence of instructions which specifies a computational procedure solving every given instance $\mathcal{I}$ of Π. Our focus in this thesis is to develop *efficient*, or *polynomial-time* algorithms. The term *efficiency* refers here to the running time of the algorithm, defined as the number of elementary operations needed to solve any instance of a given problem.

The running time of an algorithm is expressed as a function of the *size* of the input, defined formally as the number of bits needed to encode it. For example,

the size to encode an integer $a \in \mathbb{N}$ is $\lceil \log_2(a+1) \rceil$. We will often consider problems which take a graph $G = (V, E)$ as an input. Note that it is a priori unclear what the size of the instance is, since there may be different ways to encode a graph. Here are two ways possible to encode a graph $G = (V, E)$ with n nodes and m edges:

- We can store a $n \times n$ adjacency matrix $A = (a_{ij})$ with $a_{ij} = 1$ if $(i, j) \in E$ and 0 otherwise. In that case, the size of the instance is n^2.
- Another more efficient way is to maintain *adjacency lists*. For every node v in V, we maintain a list $L_v \subseteq V$ of the nodes incident to it. Observe that each edge occurs in two lists, meaning that the size is now $n + 2m$.

This illustrates the fact that the size of the instance may depend on the underlying *data structure* used to store it.

To simplify the analysis of the running time and avoid dependencies on unimportant details, we are often not interested in the exact number of elementary operations, but only in their *order*. This means that we are only interested in determining the running time of an algorithm up to constant factors. The Landau O, Ω and Θ notations are commonly used for relating expressions of the same order and are defined as follows.

Definition 2.1.1. Let $g : \mathbb{N} \to \mathbb{R}_{\geq 0}$ be a function. We write $f = O(g)$ if

$$\exists c > 0, n_0 \in \mathbb{N} \text{ such that } f(n) \leq c\, g(n) \quad \forall n \geq n_0.$$

Moreover, we write $f = \Omega(g)$ if $g = O(f)$, and we write $f = \Theta(g)$ if $f = O(g)$ and $f = \Omega(g)$ simultaneously.

We are now ready to define efficient algorithms and polynomial-time solvable problems.

Definition 2.1.2. An algorithm is *efficient*, i.e. runs in *polynomial time* if its running time $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is bounded by a polynomial in the size of the input, i.e. there is a polynomial g such that $f = O(g)$. A computational problem is *solvable in polynomial time* if there exists an efficient algorithm which solves it for any given instance.

The three problems introduced in the previous section are in fact examples of polynomial-time solvable problems. The minimum spanning tree problem can be solved in time $O(m + n \log(n))$, the shortest path problem can be solved in time $O(n+m)$ and the maximum weight matching problem can be solved in time $O(n^2 m)$.

2.1.2 Complexity classes P and NP

There exist however many fundamental optimization problems for which efficient algorithms have not been found. Here is an example of such a problem.

Minimum weight vertex cover. The input is a graph $G = (V, E)$ with a weight function $w : V \to \mathbb{R}_{\geq 0}$. The goal is to find a subset of vertices $S \subseteq V$ covering all the edges of the graph, i.e. $|S \cap (u, v)| \geq 1$ for every $(u, v) \in E$, of minimum weight $w(S) := \sum_{v \in S} w_v$.

On the other hand, nobody has also been able to disprove that no polynomial-time algorithm exists for vertex cover. Is this problem (among others) then intrinsically more difficult than the polynomial-time problems mentioned previously? *Complexity theory* aims to understand this question better.

One usually then considers decision problems, which are algorithmic problems whose output needs to be a *yes* or *no* answer. Here are two examples of such problems.

- **Prime.** The input is a natural number $n \in \mathbb{N}$. The goal is to determine whether n is prime.
- **Graph connectedness.** The input is a graph $G = (V, E)$. The goal is to determine whether G is connected.

There is a natural way to convert any optimization problem into a decision problem. Let Π be an optimization problem with ground set N, feasible solutions $\mathcal{F} \subseteq 2^N$ and an objective function $w : \mathcal{F} \to \mathbb{Q}$ to minimize. We can add a parameter $k \in \mathbb{N}$ to the input and ask the following algorithmic question: *does there exist a feasible solution $S \in \mathcal{F}$ with objective value $w(S) \leq k$?* Note that this indeed becomes a decision problem, since the goal of an algorithm is simply to output *yes* or *no*. Clearly, the decision version is an easier problem. However, having an efficient algorithm for the decision version would (in most cases) also imply an efficient algorithm for the optimization version, by simply doing a binary search on k to find the optimal solution.

We now define the complexity class P, which stands for *polynomial-time*.

Definition 2.1.3. A decision problem Π belongs to the complexity class P if there exists an efficient (polynomial-time) algorithm which, for every instance $\mathcal{I}$ of Π determines in polynomial time whether $\mathcal{I}$ is a *yes* or a *no* instance.

We can now also define the complexity class NP, which stands for *non-deterministic polynomial time*. This is the class which contains all the decision problems for which *yes* instances can be *verified* by a polynomial-size certificate in polynomial time. For the decision version of an optimization problem, we say that S is a certificate if $S \in \mathcal{F}$ and $w(S) \leq k$.

Definition 2.1.4. A decision problem Π belongs to the complexity class NP if every *yes* instance $\mathcal{I}$ admits a certificate whose validity can be verified in polynomial time.

2.1.3 NP-Hard problems

It is clear that $P \subseteq NP$. Several decades of research seem to suggest that there are problems in NP which are intrinsically harder than the ones in P. However, the question of whether $P \neq NP$ is still one of the biggest mysteries in mathematics and computer science: it constitutes one of the seven millennium-prize problems.

In complexity theory, a computational problem Π is called NP-hard if, for every problem L in NP, there is a polynomial time reduction from L to Π. Informally, this means that, if a solution for Π can be found in polynomial time, then the solution to Π can be used to solve L in polynomial time. An efficient algorithm for an NP-hard problem would then imply that there exists an efficient algorithm to solve *any* problem in NP, and it is thus very unlikely that one exists. *Vertex cover* is one example of such a problem. Here are additional examples of NP-hard optimization problems of special interest in this thesis.

Maximum k-set packing. The input is a hypergraph $\mathcal{H} = (V, H)$, with a set of nodes V and a collection $H \subseteq 2^V$ of *hyperedges*, where each hyperedge $h \in H$ has cardinality $k \geq 2$. The goal is to find a disjoint subset of hyperedges $M \subseteq H$ of maximum cardinality. This problem is also sometimes called *maximum matching on k-uniform hypergraphs*.

Load balancing. The input is a set of jobs J and a set of machines M. Each job $j \in J$ has a weight $w_{ij} \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ on machine $i \in M$. The goal is to find an assignment of jobs to machines to minimize the sum of squares of the loads of the machines $\sum_i L_i^2$, where L_i denotes the total weight of jobs assigned to that machine.

Scheduling on unrelated machines. The input is a set of jobs J and a set of machines M. Each job $j \in J$ has a weight $w_j \in \mathbb{R}_{\geq 0}$ and a processing time $p_{ij} \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ on machine $i \in M$. The goal is to find an assignment of jobs to machines, along with an ordering of jobs on each machine, in order to minimize the sum of weighted completion times of the jobs.

Even though these problems are NP-hard, some special cases of them can be polynomial-time solvable. For instance, the minimum weight vertex cover problem can be solved efficiently on bipartite graphs. The maximum k-set packing problem with $k = 2$ is simply the maximum cardinality (or unweighted, meaning

that $w = \mathbb{1}$) matching problem. Unweighted scheduling on unrelated machines (again meaning that $w = \mathbb{1}$) can also be solved in polynomial-time.

2.1.4 Integer programming formulations

Very often, combinatorial optimization problems can be formulated as an integer program (or IP), which can be cast in general as the following optimization problem:

$$\min_x \left\{ f(x) : Ax \geq b,\ x \in \mathbb{Z}^n_{\geq 0} \right\}$$

where $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^n$ and where we assume in this thesis that $f : \mathbb{R}^n \to \mathbb{R}$ is either *linear* or *quadratic* in x. Let us illustrate two examples of how NP-hard problems described in the previous section can be modeled as an IP.

The minimum weight vertex cover problem can be modeled as an integer program with a *linear* objective function. The constraints of the program encode the fact that each edge must have at least one of its endpoints picked in the cover:

$$\begin{aligned} \min \sum_{v \in V} w_v\, x_v & \qquad (2.1.1) \\ x_u + x_v \geq 1 & \qquad \forall (u,v) \in E \\ x_v \in \{0,1\} & \qquad \forall v \in V. \end{aligned}$$

The load balancing problem can also be modeled as an integer program, but with a *quadratic* objective function. The constraints encode the fact that each job must be assigned to exactly one machine:

$$\begin{aligned} \min \sum_{i \in M} \Big(\sum_{j \in J} x_{ij}\, w_{ij} \Big)^2 & \qquad (2.1.2) \\ \sum_{i \in M} x_{ij} = 1 & \qquad \forall j \in J \\ x_{ij} \in \{0,1\} & \qquad \forall j \in J, \forall i \in M. \end{aligned}$$

It turns out that the maximum k-set packing can be modeled as an IP with a linear objective function, whereas scheduling on unrelated machines can be cast as an IP with a quadratic objective function. We leave the exact description of these programs to later chapters in this thesis.

Solving these integer programs is of course still NP-hard. However, these formulations will later allow us to formulate *convex relaxations*, such as *linear* or *semidefinite programs*, which can in fact be solved in polynomial time. We describe this in more detail in Section 2.3 and Section 2.4.

2.2 Approximation algorithms

For a given NP-optimization problem Π and an instance $\mathcal{I}$ of Π, let us denote by $\mathsf{OPT}(\mathcal{I})$ the objective value of an optimal solution. When the instance is clear from the context, we will sometimes just write OPT. For NP-hard problems, since finding an optimal solution is a difficult task, one often wants to find a good approximate solution in polynomial time, leading to the following definition.

Definition 2.2.1. An *approximation algorithm* for a minimization problem Π with *approximation ratio* $\alpha \geq 1$ is a polynomial-time algorithm which, for any instance $\mathcal{I}$ of Π, returns a feasible solution with objective value at most $\alpha\mathsf{OPT}(\mathcal{I})$.

This definition can be defined analogously for maximization problems. Let us now illustrate this definition with a simple example of an approximation algorithm for the minimum (unweighted) vertex cover problem, which is NP-hard.

Algorithm 2.2.1

Input: Graph $\mathcal{G} = (V, E)$
Output: Vertex cover $S \subseteq V$

1: Compute a *maximal* (under inclusion) matching $M \subseteq E$
2: **return** $S := \{v \in V : v \text{ is matched by } M\}$

Theorem 2.2.2. *The above algorithm is a 2-approximation algorithm for the unweighted minimum vertex cover problem.*

Proof:
Observe first that this is indeed a polynomial-time algorithm: a maximal matching can be computed by greedily picking an available edge, removing its endpoints, and repeating this process. Observe also that it indeed returns a feasible solution: if an edge is left uncovered by S, it would contradict the maximality of M. Finally, observe that $|M| \leq \mathsf{OPT}$, since any feasible vertex cover needs to pick at least one endpoint of every edge in M, leading to $|S| = 2|M| \leq 2\,\mathsf{OPT}$. □

2.3 Linear programming

An extremely powerful tool in the design of approximation algorithms is linear programming. A linear program (or LP) is the problem of optimizing a linear function subject to linear inequality constraints, a task which turns out to be efficiently solvable. A minimization LP in canonical form can be written as:

$$\begin{aligned} \min\ & c^T x \\ & Ax \geq b \\ & x \geq 0 \end{aligned} \tag{2.3.1}$$

where $A \in \mathbb{Q}^{m\times n}, b \in \mathbb{Q}^m, c \in \mathbb{Q}^n$ and $n, m \in \mathbb{N}$ is the *input data*. The *variables* of this program are the entries of the vector $x \in \mathbb{R}^n$. This is thus a linear program with n variables and m constraints.

2.3.1 Duality

Linear programming admits a very important duality theory. Every linear program has a corresponding closely related dual linear program. The dual to (2.3.1) is the following:

$$\begin{aligned} \max\ & b^T y \\ & A^T y \leq c \\ & y \geq 0. \end{aligned} \tag{2.3.2}$$

Let us call the LP (2.3.1) to be the *primal* problem. The dual problem now has m variables and n constraints. Each constraint in the primal corresponds to a variable in the dual, and each variable in the primal corresponds to a constraint in the dual.

These two programs satisfy *weak duality*, which states that the objective value of any feasible dual solution y is a lower bound on the objective value of any feasible primal solution x, i.e. $b^T y \leq c^T x$. Moreover, if the optimal objective value of both of these programs is finite, then strong duality holds, meaning that $b^T y^* = c^T x^*$ for any optimal primal and dual solutions x^* and y^*. In addition, (x, y) is an optimal primal-dual pair if and only if the following *complementary slackness conditions* are satisfied:

1. $x_i > 0 \implies (A^T y)_i = c_i$
2. $y_j > 0 \implies (Ax)_j = b_j$.

This says that whenever a variable is strictly positive, the corresponding constraint in the dual program is *tight*.

2.3.2 Extreme points

The feasible region of a linear program is a finite intersection of halfspaces and is called a *polyhedron*. If a polyhedron is bounded, it is also called a *polytope*. For instance, the feasible region for the LP (2.3.1) is the polyhedron $P := \{x \in \mathbb{R}^n_{\geq 0} : Ax \geq b\}$. The "corners" of this region are called the extreme points and are formally defined as follows.

Definition 2.3.1. A point $x \in P$ is an *extreme point* of P if it is not the midpoint of two other points in P, i.e. there does not exist $y, z \in P$ with $y \neq z$ such that $(y + z)/2 = x$.

It turns out that, if the optimal LP solution has finite value, then there always exists an optimal extreme point solution. In fact, a commonly used algorithm in practice to solve LPs called the *simplex method* iteratively moves from extreme point to extreme point while trying to improve the objective value of the solution.

2.3.3 Relax and round paradigm

Linear programming can be used to design approximation algorithms for combinatorial optimization problems using the following general paradigm. First, an integer program (IP) of the problem is formulated. Since an IP is NP-hard to solve, one can relax the integrality constraints to obtain an LP, which can then be solved efficiently to obtain an optimal *fractional* solution. This fractional solution is then rounded into a feasible integer solution while preserving some guarantees on the objective value.

Let us illustrate this paradigm for the weighted vertex cover problem, whose IP we formulated in (2.1.1). Relaxing the integrality constraints gives us a linear program:

$$\begin{aligned} \min \sum_{v \in V} w_v \, x_v & \qquad (2.3.3) \\ x_u + x_v \geq 1 & \qquad \forall (u,v) \in E \\ x_v \geq 0 & \qquad \forall v \in V. \end{aligned}$$

It turns out that the feasible region of this linear program satisfies the fact that every *extreme point* $x^* \in \mathbb{R}^V$ is half-integral, meaning that $x^*_v \in \{0, \frac{1}{2}, 1\}$ for every $v \in V$. This structural understanding leads to the following 2-approximation algorithm for this problem.

Algorithm 2.3.1

Input: Weighted graph $\mathcal{G} = (V, E, w)$
Output: Vertex cover $S \subseteq V$

1: Compute an optimal extreme point solution x^* to the LP (2.3.3)
2: **return** $S := \{v \in V : x^*_v \geq 1/2\}$

Theorem 2.3.2. *The above algorithm is a 2-approximation algorithm for the weighted minimum vertex cover problem.*

Proof:
Note first that the algorithm indeed returns a feasible vertex cover: for every edge $(u, v) \in E$, one of the LP constraints tells us that $x^*_u + x^*_v \geq 1$, meaning that at least one of those those endpoints will be picked in S. To argue about the approximation guarantee, observe that $\sum_v w_v x^*_v \leq \mathsf{OPT}$, since the first term

is the optimal objective value of the LP (2.3.3), whereas the second term is the optimal objective value of the IP (2.1.1). The objective value of the returned solution is $\sum_{v \in S} w_v \leq 2 \sum_{v \in V} w_v x_v^* \leq 2\,\mathsf{OPT}$, where the first inequality follows from half-integrality of x^*. □

2.3.4 Integrality of polyhedra

We have seen in the previous section a natural way to relax an integer linear program by turning it into an efficiently solvable LP. Consider an arbitrary linear integer program of the following form:

$$\mathsf{OPT} = \min_x \left\{ c^T x : Ax \geq b,\ x \in \mathbb{Z}^n_{\geq 0} \right\}$$

where we assume that A and b are integer valued. The natural linear programming relaxation is then the following:

$$\mathsf{OPT}_{LP} = \min_x \left\{ c^T x : Ax \geq b,\ x \geq 0 \right\}. \tag{2.3.4}$$

Clearly, since the LP optimizes over a larger domain, we have that $\mathsf{OPT}_{LP} \leq \mathsf{OPT}$. We now describe a sufficient condition which guarantees that $\mathsf{OPT}_{LP} = \mathsf{OPT}$.

Theorem 2.3.3. *If the matrix A is totally unimodular (TU), meaning that all its subdeterminants lie in $\{-1, 0, 1\}$, then all the extreme points of the polyhedron $P = \{Ax \geq b, x \geq 0\}$ are integral.*

Since there always exists an optimal extreme point solution to a finite LP, the above theorem implies that $\mathsf{OPT}_{LP} = \mathsf{OPT}$ if A is totally unimodular. This is a very important observation: it means that the integer program can be solved in polynomial time by simply solving the LP relaxation.

Given a weighted graph $G = (V, E, w)$, let us take $A \in \mathbb{R}^{E \times V}$ to be the edge-vertex incidence matrix, meaning that $A_{e,v} = 1$ if and only if $v \in e$ and let us take $c = w$. Observe that in that case the LP (2.3.4) now becomes the standard weighted vertex cover LP (2.3.3). The total unimodularity of this constraint matrix is characterized by the graph structure.

Theorem 2.3.4. *The edge-vertex incidence matrix of a graph is TU if and only if the graph is bipartite.*

Therefore, this tells us that the weighted vertex cover problem is efficiently solvable on bipartite graphs: one simply needs to find an optimal extreme point solution to the LP (2.3.3), which will be guaranteed to be integral by the above two theorems.

2.4 Semidefinite programming

In this section, we describe a family of convex programs called *semidefinite programs* which generalize linear programs. To do so, we first need to introduce the following definition.

Definition 2.4.1. A symmetric matrix $X \in \mathbb{R}^{n\times n}$ is *positive semidefinite*, denoted as $X \succeq 0$, if the following equivalent conditions hold:

1. $x^T X x \geq 0$ for all $x \in \mathbb{R}^n$
2. All the eigenvalues of X are non-negative
3. There exist vectors $v_1, \ldots, v_n \in \mathbb{R}^k$ for some $k > 0$ such that $X_{ij} = \langle v_i, v_j \rangle$ for all $i, j \in [n]$.

A matrix X is *positive definite*, denoted as $X \succ 0$, if the first condition above holds with strict inequality.

We also introduce the *trace inner product*. For $A, B \in \mathbb{R}^{n\times n}$, it is defined as:

$$\langle A, B \rangle := \mathrm{Tr}(A^T B) = \sum_{i,j=1}^{n} A_{ij}\, B_{ij}.$$

The following optimization problem is then called a *semidefinite program (or SDP)*:

$$\begin{aligned} \inf & \langle C, X \rangle \\ & \langle A_i, X \rangle = b_i \quad \forall i \in [m] \\ & X \succeq 0 \end{aligned} \tag{2.4.1}$$

where the vector $b \in \mathbb{R}^m$ and the matrices C and A_i for all $i \in [m]$ are the input data. The variable of this program are the entries of the matrix X. If the matrices C and A_i are diagonal matrices with diagonals $c \in \mathbb{R}^n$ and $a_i \in \mathbb{R}^n$, then the above reduces to the linear program $\inf\{c^T x : a_i^T x = b_i \quad \forall i \in [m], x \geq 0\}$. Any linear program can in fact be written in this form, hence semidefinite programming contains linear programming as a special case.

Each SDP of the form (2.4.1) admits a dual semidefinite program

$$\begin{aligned} \sup & \sum_{i=1}^{m} y_i\, b_i \\ & \sum_{i=1}^{m} y_i\, A_i - C \succeq 0 \end{aligned} \tag{2.4.2}$$

where the variables are $y_i \in \mathbb{R}$ for all $i \in [m]$. As in linear programming, *weak duality* holds, meaning that the objective value of a feasible dual solution is always upper bounded by the objective value of a feasible primal solution. In contrast to LPs, *strong duality* does not always hold. There are however sufficient conditions for it to hold, for example if the primal program admits a positive definite matrix $X \succ 0$ as a feasible solution (also known as *Slater's condition*).

2.4.1 Relaxing quadratic integer programs

We have seen in Section 2.3.3 how LPs allow us to relax integer programs with a linear objective function. In this section, we describe how SDPs can be used to relax integer programs with a quadratic objective function. Let us illustrate this approach on the integer programming formulation of the load balancing problem (2.1.2), whose objective function we can rewrite as:

$$\sum_{i \in M} \Big(\sum_{j \in J} x_{ij}\, w_{ij} \Big)^2 = \sum_{i \in M} \sum_{j,k \in J} w_{ij}\, w_{ik}\, x_{ij}\, x_{ik}.$$

The variable of the SDP will be a matrix X with dimension $|M||N|+1$. It has one row/column corresponding to each variable x_{ij} for every pair $(i,j) \in M \times J$ of the original binary quadratic program (2.1.2), in addition to one extra row/column that we index by 0. A valid SDP relaxation of the form (2.4.1) is then given by:

$$\begin{aligned} \inf \sum_{i \in M} \sum_{j,k \in J} w_{ij}\, w_{ik}\, X_{\{ij,\, ik\}} & \\ \sum_{i \in M} X_{\{ij,\, ij\}} &= 1 && \forall j \in N \\ X_{\{0,0\}} &= 1 & & \\ X_{\{0,\, ij\}} &= X_{\{ij,\, ij\}} && \forall j \in N, i \in M \\ X &\succeq 0. & & \end{aligned}$$

To see that this is in fact a relaxation to the quadratic program (2.1.2), note that for any binary feasible assignment x, the rank-one matrix $X = (1,x)(1,x)^T$ is a feasible solution to the SDP with the same objective value. The key observation that makes this work is the fact that $x_{ij}^2 = x_{ij}$ for $x_{ij} \in \{0,1\}$, leading to $X_{\{ij,\, ij\}} = x_{ij}^2 = x_{ij} = X_{\{0,\, ij\}}$ and $X_{\{ij,\, i'k\}} = x_{ij}\, x_{i'k}$ for every $(i,j) \neq (i',k)$. Such a way of relaxing binary integer programs (and more general polynomial optimization problems) has been developed by [Las01].

2.5 Online algorithms

In online algorithms, the instance to a combinatorial optimization problem is not known fully in advance, but is revealed incrementally over time. An online

algorithm must then take an irrevocable decision each time a part of the instance is revealed. Since this definition is slightly problem dependent, we give two concrete examples of online problems.

Online bipartite matching. The input is a bipartite graph $G = (A \cup B, E)$, where A are the offline nodes and B are the online nodes. At each step, a node $v \in B$ arrives and reveals all its incident edges $\delta(v) \subseteq E$. An algorithm must take an irrevocable decision and decide which edge $e \in \delta(v)$ (if any) to add to the matching. The goal is to maximize the cardinality of the obtained matching.

Online load balancing. The input is a set of jobs J and a set of machines M. At each step, a job $j \in J$ arrives and reveals non-negative weights $\{w_{ij}\}_{i \in M}$. An online algorithm must decide irrevocably to which machine it wants to assign this job. The goal is to minimize the sum of squares of the loads of the machines $\sum_{i \in M} L_i^2$, where L_i is the total weight of jobs assigned to $i \in M$.

The quality of an online algorithm is measured by measuring the worst-case ratio (over all online instances) of the cost obtained by the algorithm to the cost of the optimal *offline* solution (i.e. after the whole instance has been revealed). This ratio is known as the *competitive ratio*. For maximization problems, we adopt in this thesis the convention that the competitive ratio is between 0 and 1. For minimization problems, we adopt the convention that it is greater or equal to 1.

Definition 2.5.1. An *online algorithm* for a minimization problem Π is $\alpha \geq 1$ *competitive* if for any online instance $\mathcal{I}$ of Π, it returns a feasible solution of cost at most $\alpha\, \mathsf{OPT}(\mathcal{I})$, where $\mathsf{OPT}(\mathcal{I})$ is the offline optimum. For a maximization problem, an online algorithm is $\alpha \in [0, 1]$ *competitive* if it returns a feasible solution of value at least $\alpha\, \mathsf{OPT}(\mathcal{I})$.

2.5.1 Example: greedy algorithm

Let us now illustrate the simplest example of an online algorithm and showcase how a suitably chosen dual solution can help us analyze it. We consider the online bipartite matching problem on a graph $G = (V, E) = (A \cup B, E)$, whose canonical primal and dual LP relaxations are given by

$$\begin{aligned} \max \sum_{e \in E} x_e & & \min \sum_{v \in V} y_v & \\ \sum_{e \in \delta(v)} x_e \leq 1 \quad & \forall v \in V & y_u + y_v \geq 1 \quad & \forall (u, v) \in E \\ x_e \geq 0 \quad & \forall e \in E & y_v \geq 0 \quad & \forall v \in V. \end{aligned}$$

Observe that the dual is simply the unweighted vertex cover LP (2.3.3).

Algorithm 2.5.1 Greedy algorithm

Input: Bipartite graph $G = (A \cup B, E)$ with online arrivals of nodes in B
Output: Matching $M \subseteq E$

set $M = \emptyset$
when $v \in B$ **arrives with** $\delta(v) \subseteq E$**:**
    pick an arbitrary edge $(u, v) \in \delta(v)$ disjoint from M if one exists
    update $M = M \cup (u, v)$
return M

Theorem 2.5.2. *The greedy algorithm (Algorithm 2.5.1) is* $1/2$*-competitive for the online bipartite matching problem.*

Proof:
Consider the following process of simultaneously building a primal and dual solution during the execution of the algorithm: whenever the algorithm picks an edge $e = (u, v) \in M$, set $x_e = 1$ and set $y_u = 1/2, y_v = 1/2$ for the two endpoints. Clearly, the objective values of the primal and dual are equal $\sum_{e \in E} x_e = \sum_{v \in V} y_v$ at all times. In addition, at the end of the algorithm, observe that for any edge $e = (u, v) \in E$, at least one of its endpoints has a dual value of $1/2$, since if this was not the case, the edge would have been picked by the greedy algorithm. Therefore, $y_u + y_v \geq 1/2$, implying that $2y$ is a feasible dual solution. By weak duality, this implies $2 \sum_{v \in V} y_v \geq \mathsf{OPT}$ and thus $|M| = \sum_{e \in E} x_e = \sum_{v \in V} y_v \geq \mathsf{OPT}/2$. □

The above analysis is an example of a primal-dual analysis for online algorithms. It can be extended in a straightforward way to the online matching problem on k-uniform hypergraphs under vertex arrivals, where each (hyper)edge $e \in E$ now has cardinality k, instead of 2. In that case, it can easily be shown that the greedy algorithm is $1/k$-competitive.

2.5.2 Fractional and randomized algorithms

Observe that the two problems (online bipartite matching and online load balancing) described at the beginning of Section 2.5 can be equivalently seen as follows. In the bipartite matching problem, whenever a node $v \in B$ arrives and reveals its incident edges $\delta(v) \subseteq E$, an (integral) online algorithm can irrevocably increase one of the variables x_e for $e \in \delta(v)$ while staying inside of the feasible region

$$\{x \in \{0, 1\}^E : x(\delta(v)) \leq 1 \quad \forall v \in A \cup B\}.$$

A *randomized* algorithm can make a random choice, i.e. pick a probability distribution over $e \in \delta(v)$ inducing random variables $X_e \in \{0, 1\}$ lying in the above

feasible region. A *fractional* algorithm can irrevocably increase the variables x_e *fractionally* while staying inside of the bipartite matching polytope

$$P_M = \{x \in [0,1]^E : x(\delta(v)) \leq 1 \quad \forall v \in A \cup B\}.$$

The value of a randomized algorithm is then measured as $\sum_{e \in E} \mathbb{E}[X_e]$. The value of a fractional algorithm is measured as $\sum_{e \in E} x_e$.

Similarly, for the load balancing problem, whenever $j \in J$ arrives and reveals the weights $\{w_{ij}\}_{i \in M}$, an integral algorithm needs to increase one of the variables x_{ij} for $i \in M$ to one while staying inside of the feasible region

$$\Big\{x \in \{0,1\}^{M \times N} : \sum_{i \in M} x_{ij} = 1 \quad \forall j \in J\Big\}.$$

A randomized algorithm can make a random choice inducing random variables $X_{ij} \in \{0,1\}$, and a fractional algorithm needs to irrevocably increase the variables x_{ij} fractionally while staying inside of the polytope

$$P_L = \Big\{x \in [0,1]^{M \times N} : \sum_{i \in M} x_{ij} = 1 \quad \forall j \in J\Big\}.$$

The cost of a randomized algorithm in this case is measured as $\sum_{i \in M} \mathbb{E}[L_i(X)^2]$, where $L_i(X) := \sum_{j \in J} w_{ij} X_{ij}$. The cost of a fractional algorithm is measured as $\sum_{i \in M} L_i(x)^2$, where $L_i(x) := \sum_{j \in J} w_{ij}\, x_{ij}$.

In both of these models, the fractional version is an easier problem, as shown in the following lemma.

Lemma 2.5.3. *Any randomized algorithm for the online bipartite matching problem induces a fractional algorithm with the same value. Similarly, any randomized algorithm for the load balancing problem induces a fractional algorithm with lower or equal cost.*

Proof:
For the bipartite matching problem, a randomized algorithm induces random variables $X_e \in \{0,1\}$ for every $e \in E$. Define a fractional algorithm which sets $x_e := \mathbb{E}[X_e]$ for every $e \in E$. Clearly, this is a valid fractional algorithm as $x \in P_M$ and the values of both algorithms match.

For the load balancing problem, a randomized algorithm induces random variables $X_{ij} \in \{0,1\}$ for every $i \in M, j \in J$. Define a fractional algorithm by setting $x_{ij} := \mathbb{E}[X_{ij}]$ for every $i \in M, j \in J$. Clearly, we then have $x \in P_L$ and

$$\sum_{i \in M} L_i(x)^2 = \sum_{i \in M} \mathbb{E}[L_i(X)]^2 \leq \sum_{i \in M} \mathbb{E}[L_i(X)^2]$$

where the first equality follows from linearity of expectation and the inequality follows from Jensen's inequality. □

2.6 Algorithmic game theory and price of anarchy

In previous sections, for a given discrete optimization problem with cost function C and an optimal solution x^*, we cared about understanding the ratio $C(x)/C(x^*)$, where x is a (non-optimal) solution obtained through an approximation algorithm or an online algorithm. In the first case, the worst-case of this ratio over all instances is called the *approximation ratio*, whereas in the second case it is called the *competitive ratio*. In this section, we consider x to be a *game theoretic equilibrium* and will care about the ratio $C(x)/C(x^*)$ in this case. The worst-case bound over all instances will now be called the *price of anarchy* of the considered game.

2.6.1 Strategic games and Nash equilibria

Definition 2.6.1. An instance $\mathcal{I}$ of a *finite strategic game*, denoted by $\Gamma = (N, (\mathcal{S}_j)_{j\in N}, (C_j)_{j\in N})$, consists of:

- A set $N = \{1, \dots n\}$ of players
- A strategy set $\mathcal{S}_j$ for every player $j \in N$
- A cost function $C_j : \mathcal{S}_1 \times \cdots \times \mathcal{S}_n \to \mathbb{R}$ for every player $j \in N$.

We will in this thesis adopt the convention that each player j wants to selfishly minimize his/her own cost function C_j.

We denote a pure assignment by a vector $\sigma = (\sigma_1, \dots, \sigma_n)$, where $\sigma_j \in \mathcal{S}_j$ for every $j \in N$. It will often be useful to have a notation where a given player j is the only one to change his/her strategy. For this reason, we denote (σ_{-j}, i) to be the vector obtained by replacing σ_j by $i \in \mathcal{S}_j$ in the jth coordinate of σ. We are now ready to define our first equilibrium concept.

Definition 2.6.2. An assignment $\sigma \in \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ is a *pure Nash equilibrium* if

$$C_j(\sigma) \leq C_j(\sigma_{-j}, i) \qquad \forall j \in N, \forall i \in \mathcal{S}_j.$$

Intuitively, a pure Nash equilibrium means that, given knowledge of the choices of the other players, no player can switch his/her strategy to get a better payoff. Unfortunately, there exist strategic games where pure Nash equilibria do not exist. One can then consider a generalization of pure Nash equlibria, called *mixed* Nash equilibria. In that case, players are allowed to independently randomize their choice over their strategy set. Each player j can thus now pick a

probability distribution $(x_{ij})_{i\in\mathcal{S}_j}$. We denote the collection of these distributions as $x = (x_{ij})_{j\in N, i\in\mathcal{S}_j}$. The expected cost of a player j is then defined as:

$$C_j(x) := \mathbb{E}_{\sigma\sim x}[C_j(\sigma)]$$

where $\sigma \sim x$ is a random assignment where each player independently randomly picks a strategy according to his/her distribution specified by x. We will refer to a collection of probability distributions $x = (x_{ij})_{j\in N, i\in\mathcal{S}_j}$ as a *mixed assignment*.

Definition 2.6.3. A mixed assignment $x = (x_{ij})_{j\in N, i\in\mathcal{S}_j}$ is a *mixed Nash equilibrium* if

$$\mathbb{E}_{\sigma\sim x}[C_j(\sigma)] \leq \mathbb{E}_{\sigma\sim x}[C_j(\sigma_{-j}, i)] \qquad \forall j \in N, \forall i \in \mathcal{S}_j.$$

It is known that any finite strategic game admits a mixed Nash equilibrium [Nas24]. Note that when x is binary, meaning that $x_{ij} \in \{0, 1\}$ for every $j \in N, i \in \mathcal{S}_j$, then it simply corresponds to an assignment $\sigma \in \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$, where $x_{ij} = 1$ indicates that player j chooses strategy i, i.e. that $\sigma_j = i$. We will thus sometimes simply refer to x as an *assignment*, meaning that it can either be a pure or a mixed assignment. We will also sometimes refer to a mixed Nash equilibrium as simply a *Nash equilibrium*.

2.6.2 Price of anarchy

We are interested in analyzing how "inefficient" in terms of cost a Nash equilibrium can be. To do so, we first need to define some global objective function, also called the *social cost*. In this thesis, we will simply care about the (possibly weighted) sum of costs incurred by the players. We note however that other definitions are possible, for instance taking the maximum cost incurred by one of the players.

Definition 2.6.4. The *social cost* of an assignment x is defined as:

$$C(x) := \sum_{j\in N} C_j(x).$$

Remark 2.6.5. For some games, we will have weights $w_j \geq 0$ for every player j and define the social cost as $C(x) := w_j C_j(x)$.

Minimizing the social cost over all pure assignments while disregarding the game-theoretical aspect is simply a discrete optimization problem. It can be formulated as the following integer program:

$$\begin{aligned} \min\, & C(x) \\ & \sum_{i\in\mathcal{S}_j} x_{ij} = 1 \qquad \forall j \in N \\ & x_{ij} \in \{0, 1\} \qquad \forall j \in N, \forall i \in \mathcal{S}_j \end{aligned}$$

where the constraints encode the fact that every player needs to pick exactly one strategy. An optimal solution to this optimization problem is known as the *social optimum*, which we will denote by x^*. We are now ready to define the quantity of interest to us.

Definition 2.6.6. The *price of anarchy* of an instance $\mathcal{I}$ of a finite strategic game Γ is defined as

$$\sup\left\{\frac{C(x)}{C(x^*)} : x \text{ is a Nash equilibrium}\right\}.$$

The price of anarchy of the game Γ is the supremum over all instances $\mathcal{I}$ of Γ of the above quantity.

2.6.3 Example: load balancing

Let us illustrate the notions defined in the previous two subsections on a concrete example: a weighted load balancing game.

We are given a set of players N and a set of resources E. The strategy set of every player $j \in N$ is a subset of resources $\mathcal{S}_j \subseteq E$ with unrelated weights $w_{ij} \geq 0$ associated for every $i \in \mathcal{S}_j$. Consider a pure assignment x, the *load* of a resource $i \in E$ is defined as the total weight of players assigned to it and is formally defined as

$$\ell_i(x) = \sum_{j \in N} w_{ij}\, x_{ij} \qquad \forall i \in E.$$

The cost of a player $j \in N$ is then defined as

$$C_j(x) = \sum_{i \in E} \ell_i(x)\, w_{ij}\, x_{ij}$$

meaning that it is the weight multiplied by the load of the resource picked. The social cost can be written as follows

$$C(x) = \sum_{j \in N} C_j(x) = \sum_{i \in E} \ell_i(x)^2. \tag{2.6.1}$$

Observe that the social optimum x^* is in fact the optimal solution to the IP (2.1.2) that we formulated previously.

We now show an example of a pure Nash equilibrium for an instance of this game and show that its price of anarchy can be as high as $1 + \phi$, where $\phi := (1+\sqrt{5})/2 \approx 1.618$ is the golden ratio and satisfies the equation $\phi^2 = 1 + \phi$. This example is due to [CFK+06].

Let $n \in \mathbb{N}$ be a natural number. The instance consists of $E = [n+1]$ resources and $N = [n]$ players. The strategy set of every player $j \in N$ is $\mathcal{S}_j = \{j, j+1\}$ and the weights are defined as $w_{1,1}, w_{2,1} = \phi$ for the first player and $w_{j,j} = 1, w_{j+1,j} = \phi$ for every $j \geq 2$.

Consider the solution where each player j picks resource j. Then the first resource has one job of weight ϕ assigned to it and each resource in $\{2, \ldots, n\}$ only has one job of weight one assigned to it. This is in fact the optimum solution x^* and it has cost $C(x^*) = \phi^2 + n - 1$.

Consider now the solution x where each player j picks resource $j + 1$. We claim that this is a pure Nash equilibrium. To see this, observe that the loads of the resources under this assignment are $\ell_1(x) = 0$ and $\ell_i(x) = \phi$ for every $i \in \{2, \ldots, n+1\}$. The payoff of every player is then $C_j(x) = \phi^2$ for every $j \geq 1$. If the first player were to switch to resource 1, he would get a payoff of ϕ^2, which would not be an improvement. If a player $j \geq 2$ were to switch to resource j, then he would get a payoff of $1+\phi$, which is not an improvement since the golden ratio satisfies $1 + \phi = \phi^2$. Hence, x is a pure Nash equilibrium whose cost is $C(x) = \sum_{j \in N} C_j(x) = n\,\phi^2$. As n grows, the price of anarchy of this instance becomes:

$$\frac{C(x)}{C(x^*)} = \frac{n\phi^2}{n - 1 + \phi^2} \xrightarrow{n \to \infty} \phi^2.$$

This example shows that the price of anarchy of the load balancing game considered is *at least* $\phi^2 = 1 + \phi = (3 + \sqrt{5})/2 \approx 2.618$. In Chapter 5, we will show that this is in fact the right answer by providing a technique proving tight *upper bounds* on the price of anarchy for a large class of games similar to the one considered here.

Chapter 3

Round and bipartize for vertex cover approximation

In this chapter, we study the weighted vertex cover in a beyond the worst-case setting. We have seen a 2-approximation algorithm based on rounding the natural LP relaxation in Algorithm 2.3.1 and the fact that this LP is integral for bipartite graphs in Section 2.3.4. The question tackled here is roughly the following: *can meaningful guarantees be obtained which somehow interpolate between these two extremes?*

Given the knowledge of an induced bipartite subgraph of the input, we consider a natural rounding algorithm based on the standard LP and tightly characterize its approximation ratio. As a byproduct, we also provide tight bounds on the integrality gap of the standard LP for three colorable graphs.

3.1 Introduction

In the vertex cover problem we are given a weighted graph $\mathcal{G} = (V, E, w)$, where $w : V \mapsto \mathbb{R}_+$ is a non-negative weight function on the vertices, and the goal is to find a minimal weight subset of vertices $U \subset V$ that covers every edge of the graph, i.e.,

$$\min\{w(U) \mid U \subset V,\ |U \cap \{i, j\}| \geq 1\ \forall (i, j) \in E\}.$$

We denote by OPT an optimal subset of vertices for this problem, and by $w(\mathsf{OPT})$ the total weight of that solution. The vertex cover problem is known to be NP-complete [Kar72] and APX-complete [PY88]. Moreover, it was shown to be NP-hard to approximate within a factor of 7/6 in [Hås01], a factor later improved to 1.36 in [DS05]. It is in fact NP-hard to approximate within a factor of $2 - \varepsilon$ for any fixed $\varepsilon > 0$ if the unique games conjecture is true [KR08].

A natural linear programming relaxation, as well as its dual, is given by:

$$\begin{aligned} \min \sum_{v \in V} w_v x_v & & \max \sum_{e \in E} y_e & \\ x_u + x_v \geq 1 \quad & \forall (u,v) \in E & y(\delta(v)) \leq w_v \quad & \forall v \in V \\ x_v \geq 0 \quad & \forall v \in V & y_e \geq 0 \quad & \forall e \in E \end{aligned}$$

For a given graph $\mathcal{G}$, we denote the primal linear program by $P(\mathcal{G})$ and the dual by $D(\mathcal{G})$. The integrality gap of the standard linear relaxation $P(\mathcal{G})$ on a graph of n vertices is upper bounded by $2 - 2/n$, a bound which is attained on the complete graph. In fact, a more fine-grained analysis shows that it is equal to $2-2/\chi^f(\mathcal{G})$, where $\chi^f(\mathcal{G})$ is the fractional chromatic number of the graph [Sin19]. An integrality gap of $2-\varepsilon$ is proved for a large class of linear programs in [ABL02]. It is also known that any linear program which approximates vertex cover within a factor of $2 - \varepsilon$ requires super-polynomially many inequalities [BFPS19].

An important property of $P(\mathcal{G})$ is the fact that any extreme point solution x^* is half-integral, i.e., $x^*_v \in \{0, \frac{1}{2}, 1\}$ for any vertex $v \in V$ [NT75]. This gives rise to a straightforward rounding algorithm by solving $P(\mathcal{G})$ and outputting all vertices whose LP variable is at least a half, i.e., $U := \{v \in V \mid x^*_v \geq \frac{1}{2}\}$. It is easy to see that this a 2-approximation, because $w(U) \leq 2\, w(\mathsf{OPT})$, see [Hoc82]. Moreover, it is known that $P(\mathcal{G})$ is integral for any bipartite graph [Kuh55]. As a consequence, the rounding algorithm returns an optimal solution if the graph is bipartite.

3.2 Outline

Set-up and algorithm

In this chapter, we initiate a beyond the worst-case study of the vertex cover problem and of its standard linear programming relaxation. As mentioned above, this simple and natural linear program is extremely powerful for this problem: it leads to the optimal approximation ratio of 2, under the unique games conjecture, and its polyhedron has some very nice structural properties since it is integral for bipartite graphs and half-integral for general graphs. We are in this chapter interested in having a more fine-grained understanding of it and see whether one can introduce parameters that allow to somehow interpolate the rounding curve of this LP in a beyond the worst-case manner.

We consider the following setup. We are given a weighted non-bipartite graph $\mathcal{G} = (V, E, w)$ and an optimal solution $x^* \in \{0, \frac{1}{2}, 1\}$ to $P(\mathcal{G})$. We denote by $V_\alpha := \{v \in V \mid x^*_v = \alpha\}$ the vertices taking value α and by $\mathcal{G}_\alpha = \mathcal{G}[V_\alpha]$ the subgraph of $\mathcal{G}$ induced by the vertices V_α for any $\alpha \in \{0, \frac{1}{2}, 1\}$. By a standard preprocessing step, we may assume that we only work on the graph $\mathcal{G}_{1/2}$, since any c-approximate solution on this reduced graph can be lifted to a c-approximate solution on the original graph by adding the nodes V_1 to the solution [NT75]. In

addition, we suppose that we have knowledge of an odd cycle transversal S of $\mathcal{G}_{1/2}$, meaning that $\mathcal{G}_{1/2} \setminus S$ is a bipartite graph. Equivalently, S intersects every odd cycle of $\mathcal{G}_{1/2}$. The question of finding a good such odd cycle transversal is also tackled later on.

We consider the following simple algorithm, detailed in Algorithm 3.2.1. It first solves $P(\mathcal{G})$, takes the vertices assigned value one by the linear program to the solution and removes all the integral nodes from the graph to arrive at $\mathcal{G}_{1/2}$. The algorithm then takes all the vertices in the set S to the solution, removes them from the graph and solves another (now integral) linear program to get the optimal solution on the bipartite remainder. These vertices are then also added to the solution.

Algorithm 3.2.1

Input: Weighted graph $\mathcal{G} = (V, E, w)$, odd cycle transversal $S \subset V_{1/2}$
Output: Vertex cover $U \subset V$

1: Solve the linear program $P(\mathcal{G})$ to get V_0 , $V_{1/2}$ and V_1
2: Solve the integral linear program $P(\mathcal{G}_{1/2} \setminus S)$ to get $W \subset V_{1/2}$
3: **return** $V_1 \cup S \cup W$

The main question studied is the following. What is the worst-case approximation ratio of the algorithm and which weight functions are attaining it?

Our motivation to study this question comes from the structural difference between the polyhedron of $P(\mathcal{G})$ for bipartite and non-bipartite graphs. In particular, we are interested in identifying parameters of the problem that enable us to derive more fine-grained bounds determining the approximation ratio of the algorithm, and allow to interpolate the rounding curve of the standard linear program from 1 to 2, depending on how far the graph is from being bipartite.

As it turns out, the *odd girth*, i.e., the length of the shortest odd cycle, is a key parameter determining tight bounds on the approximation ratio. It is also a natural parameter, since a graph is bipartite if and only if it does not contain an odd cycle. The larger the odd girth, the closer the graph is to being bipartite. It is also shown in [GKŁ97] that graphs with a large odd girth admit a small-cardinality odd cycle transversal.

Contributions and high-level view

We first do a pre-processing step and show that we may without loss of generality focus on weighted graphs $\mathcal{G} = (V, E, w)$ where the weights come from a certain weight space Q^W. Each edge has a dual weight $y_e \geq 0$ with a total sum of $y(E) = 1$, and the weight on each node is then determined by $w_v = y(\delta(v))$. This follows from the Nemhauser-Trotter theorem, complementary slackness and an appropriate normalization.

We then do the analysis under the assumption that S is a stable set, highlighting the main ideas of the analysis and the proof techniques. We show that the approximation ratio is upper bounded by $1 + 1/\rho$, where $2\rho - 1$ denotes the odd girth of the graph $\tilde{\mathcal{G}} := \mathcal{G}/S$, where all the vertices in S are contracted into a single node. Note that the parameter range is $\rho \in [2, \infty]$, with $\rho = \infty$ naturally corresponding to the case where $\tilde{\mathcal{G}}$ is bipartite. The proof technique involves a key concept, that we call *pairwise edge-separate* feasible vertex covers. Constructing k such covers allows to bound the approximation ratio by $1 + 1/k$. The construction of ρ such covers to get the result follows from a structural understanding of the contracted graph $\tilde{\mathcal{G}}$. As a byproduct, this structural understanding also allows to get improved bounds on the integrality gap and the fractional chromatic number of 3-colorable graphs. In particular, it even manages to compute an exact formula, depending on the odd girth, for the integrality gap and the fractional chromatic number of the contracted graph $\tilde{\mathcal{G}}$.

We then construct a class of weight functions $\mathcal{W} \subset Q^W$ where this upper bound holds with equality, thus showing that this proof technique obtains tight bounds and might have additional applications. This result can then be lifted to the case where S is a general set, by introducing an additional parameter α counting the total dual sum of the weights on the edges inside S, i.e. $\alpha = y(E[S])$. This then leads to an approximation ratio interpolating the rounding curve of the standard linear program with a tight bound of $(1+1/\rho)(1-\alpha)+2\alpha$ for any values of $\rho \in [2, \infty]$ and $\alpha \in [0, 1]$.

We then discuss algorithmic applications to find good such sets S and show that our analysis is optimal in the following sense: the worst case bounds for ρ and α, which are $\rho = 2$ and $\alpha = 1 - 4/n$, recover the integrality gap of $2 - 2/n$ of the standard linear programming relaxation for a graph on n vertices.

Implications and related work

Our analysis falls into the framework of beyond the worst-case analysis [Rou21]. In particular, note that an odd cycle transversal always exists: we may simply take $S = V_{1/2}$, which recovers the standard 2-approximation algorithm for vertex cover. Depending on how S is chosen, our algorithm can however admit significantly better approximation ratios.

Our algorithm also connects to *learning-augmented algorithms*, which have access to some prediction in their input (e.g., obtained for instance through machine learning). This prediction is assumed to come without any worst-case guarantees, and the goal is then to take advantage of it by making the algorithm perform better when this prediction is good, while still keeping robust worst-case guarantees when it is off [BMS20, LV21, PSK18, LLMV20, ACE+20, AGKK20]. In our case, assuming a prediction on the set S, robustness is guaranteed since we are never worse than a 2-approximation. In fact, even if the predicted set is not an odd cycle transversal, one may simply greedily add vertices to it until it becomes

one, while still guaranteeing a 2-approximation. Otherwise, our results provide a precise understanding of how the approximation ratio improves depending on the predicted set S. In addition, once such a set S is found, the parameters α and ρ can easily be computed to see the improved guarantee on the approximation ratio.

The odd cycle transversal number $\text{oct}(\mathcal{G})$ is defined as the minimum number of vertices needed to be removed in order to make the graph bipartite. The minimum odd cycle transversal problem has been studied in terms of fixed-parameter tractability [RSV04, KW14]. In particular, it is the first problem where the iterated compression technique has been applied [RSV04], now a classical tool in the design of fixed-parameter tractable algorithms. The best known approximation algorithm for it has a ratio of $O(\sqrt{\log(n)})$ [ACMM05]. Another relevant concept is the *odd cycle packing number* $\text{ocp}(\mathcal{G})$, defined as the maximum number of vertex-disjoint odd cycles of $\mathcal{G}$ and satisfying $\text{ocp}(\mathcal{G}) \leq \text{oct}(\mathcal{G})$. The related maximum stable set problem has been studied in terms of $\text{ocp}(\mathcal{G})$ in [BFMRV14, AWZ17, CFH$^+$20, FJWY22].

A key property implying the integrality of a polyhedron is the *total unimodularity* (TU) of the constraint matrix describing the underlying problem, meaning that all the square subdeterminants of the matrix are required to lie in $\{-1, 0, 1\}$ (see for instance [Sch98, Sch03]). In general, we believe it is an interesting question to study whether one may exploit the knowledge of a TU substructure in an integer program to obtain improved approximation guarantees through rounding algorithms. In our case, the knowledge of an odd cycle transveral S is equivalent to the knowledge of an induced bipartite subgraph $\mathcal{G}' = \mathcal{G} \setminus S$, for which $P(\mathcal{G}')$ is TU. We hope the techniques introduced for the pair $(\mathcal{G}, S)$ can help tackle other problems.

One technique which might also benefit from our analysis is iterative rounding, which requires solving a linear program at each iteration [LRS11]. Having a better analysis for the case where the linear program becomes integral could potentially be used to reduce the number of iterations and allow for better guarantees, since iterative rounding can terminate at this step without losing solution quality.

Several different algorithms achieving approximation ratios of $2 - o(1)$ have been found for the weighted and unweighted versions of the vertex cover problem: [Kar05, Hal02, BYE83, MS85, BYE81, Hoc83]. Another large body of work is interested in exact fixed parameter tractable algorithms for the decision version: [BG93, BFR98, CKJ01, CLJ00, DF92, NR99, NR03, SF99, DF12].

3.3 Preliminaries

We define $\mathbb{R}_+$ to be the non-negative real numbers and $[k] := \{1, \ldots, k\}$ to be the natural numbers up to $k \in \mathbb{N}$. For a vector $x \in \mathbb{R}^n$, we denote the sum of the coordinates on a subset by $x(A) := \sum_{i \in A} x_i$. A key property of $P(\mathcal{G})$ was

introduced by Nemhauser and Trotter in [NT75]. It essentially allows to reduce an optimal solution $x^* \in \{0, \frac{1}{2}, 1\}^V$ to a fully half-integral solution by looking at the subgraph induced by the half-integral vertices. As before, we denote by $V_\alpha := \{v \in V \mid x^*_v = \alpha\}$ the vertices taking value α and by $\mathcal{G}_\alpha = \mathcal{G}[V_\alpha]$ the subgraph induced by the vertices V_α.

Theorem. 3.3.1 (Nemhauser, Trotter [NT75]). *Let $x^* \in \{0, \frac{1}{2}, 1\}^V$ be an optimal extreme point solution to $P(\mathcal{G})$. Then, $w(\mathsf{OPT}(\mathcal{G}_{1/2})) = w(\mathsf{OPT}(\mathcal{G})) - w(V_1)$.*

Corollary 3.3.2. *Let $x^* \in \{0, \frac{1}{2}, 1\}^V$ be an optimal solution to $P(\mathcal{G})$. If $U \subset V_{1/2}$ is a feasible vertex cover on $\mathcal{G}_{1/2}$ with approximation ratio at most ϕ, i.e., $w(U) \leq \phi\ w(\mathsf{OPT}(\mathcal{G}_{1/2}))$, then $w(U) + w(V_1) \leq \phi\, w(\mathsf{OPT}(\mathcal{G}))$.*

Proof:
The proof is an easy consequence of Theorem 3.3.1 and the fact that $\phi \geq 1$:

$$w(U) + w(V_1) \leq \phi\, w(\mathsf{OPT}(\mathcal{G}_{1/2})) + w(V_1) \leq \phi\, w(\mathsf{OPT}(\mathcal{G})).$$

□

The previous corollary thus implies that we may restrict our attention to the graph $\mathcal{G}_{1/2}$, since any ϕ-approximate solution on this reduced instance can be lifted to a ϕ-approximate solution on the original graph by adding V_1 to the solution. Note that on the weighted graph $\mathcal{G}_{1/2}$, the solution $(\frac{1}{2}, \ldots, \frac{1}{2})$ is optimal.

For a given set $S \subset V$, we define $\mathcal{G}' := \mathcal{G} \backslash S = (V', E')$ to be the graph obtained by removing the set S and all the incident edges to it. Hence, $E = E' \cup \delta(S) \cup E[S]$ where $\delta(S) = \{(u, v) \in E \mid u \in S, v \notin S\}$ and $E[S] := \{(u, v) \in E \mid u \in S, v \in S\}$. We also denote by $\tilde{\mathcal{G}} := \mathcal{G}/S = (\tilde{V}, \tilde{E})$ the graph obtained by contracting all the vertices in S into a single new node $v^S \in \tilde{V}$. We allow for multiple edges, but no self-loops. The only edges present in E but not in $\tilde{E}$ are thus the ones with both endpoints in S, i.e., $E[S]$.

3.4 Weight space

By Corollary 3.3.2, we may assume that every weighted graph $\mathcal{G}$ we work with has the property that the fully half-integral solution $x = (\frac{1}{2}, \ldots, \frac{1}{2})$ is an optimal solution to the linear program $P(\mathcal{G})$. In this section, we characterize the weight functions satisfying this assumption.

Lemma 3.4.1. *Let $\mathcal{G} = (V, E)$ be a graph and let $w : V \to \mathbb{R}_+$ be a weight function. The feasible solution $(\frac{1}{2}, \ldots, \frac{1}{2})$ to the linear program $P(\mathcal{G})$ is optimal if and only if there exists $y \in \mathbb{R}_+^E$ satisfying $y(\delta(v)) = w_v$ for every $v \in V$.*

Proof:
By complementary slackness, a feasible solution $x \in \mathbb{R}_+^V$ to the primal $P(\mathcal{G})$ and a feasible solution $y \in \mathbb{R}_+^E$ to the dual $D(\mathcal{G})$ are optimal if and only if:

$$x_v > 0 \implies y(\delta(v)) = w_v \quad \forall v \in V$$

and

$$y_e > 0 \implies x_u + x_v = 1 \quad \forall e = (u, v) \in E.$$

If $x = (\frac{1}{2}, \ldots, \frac{1}{2})$ is an optimal solution, then, by strong duality, there exists an optimal dual solution y and this solution needs to satisfy $y(\delta(v)) = w_v$ for every $v \in V$. Conversely, if there exists a dual solution y satisfying $y(\delta(v)) = w_v$ for every $v \in V$, then the pair $x = (\frac{1}{2}, \ldots, \frac{1}{2})$ and y satisfy both the conditions of complementary slackness, implying that x is optimal. □

Such instances have been called *edge-induced* in [CFH+20, FJWY22], in the sense that the dual values on the edges are free parameters, and the weights on the nodes are determined once the dual values are fixed. Such instances also satisfy:

$$w(V) = \sum_{v \in V} w_v = \sum_{v \in V} y(\delta(v)) = \sum_{v \in V} \sum_{e \in E} y_e \, 1_{\{e \in \delta(v)\}} = \sum_{e \in E} y_e \sum_{v \in V} 1_{\{e \in \delta(v)\}} = 2\, y(E).$$

Observe that the approximation ratio of a feasible solution $U \subset V$ is defined as $w(U)/w(\mathsf{OPT}(\mathcal{G}))$ and is invariant under scaling of the weights. We thus make a normalization ensuring that the optimal LP solution has objective value one, i.e., $w(V)/2 = y(E) = 1$, to get the following weight space polytope:

$$Q^W := \big\{ w \in \mathbb{R}_+^V \mid \exists y \in [0,1]^E \text{ such that } y(E) = 1 \text{ and } w_v = y(\delta(v)) \quad \forall v \in V \big\}.$$

It turns out that the vertices of the polytope Q^W have a one-to-one correspondance with the edges of the graph. We denote by $\mathbb{1}_v \in \mathbb{R}^V$ the indicator vector of a vertex $v \in V$.

Theorem 3.4.2. *The polytope Q^W of a graph $\mathcal{G} = (V, E)$ can be expressed as:*

$$Q^W = \operatorname{conv}\Big(\Big\{ \mathbb{1}_u + \mathbb{1}_v \;\Big|\; (u, v) \in E \Big\} \Big).$$

Moreover, $\mathbb{1}_u + \mathbb{1}_v$ is an extreme point of Q^W for every edge $(u, v) \in E$.

Proof:
Let $w \in Q^W$ and $y \in \mathbb{R}_+^E$ such that $y(\delta(v)) = w_v$ for every $v \in V$, as well as $y(E) = 1$. Observe that:

$$w = \sum_{(u,v) \in E} y_{(u,v)} \big(\mathbb{1}_u + \mathbb{1}_v \big).$$

By looking at this equality coordinate by coordinate, for a fixed vertex $v \in V$, the contribution from the left hand side is w_v whereas the contribution from the right hand side is $y(\delta(v))$. We have thus managed to decompose an arbitrary $w \in Q^W$ into a convex combination of the vectors $\left\{\mathbb{1}_u + \mathbb{1}_v \;\middle|\; (u,v) \in E\right\}$.

Let $(u,v) \in E$ and let us now show that $\bar{w} := \mathbb{1}_u + \mathbb{1}_v$ is an extreme point of Q^W. Firstly, it is clear that $\bar{w} \in Q^W$, the dual solution satisfying complementary slackness being $y_{(u,v)} = 1$ and $y_e = 0$ for every $e \in E \setminus (u,v)$, and the sum of all the weights being indeed equal to two. Suppose for contradiction that there exist distinct $w^1, w^2 \in Q^W$ such that $\bar{w} = \frac{1}{2}(w^1 + w^2)$. Notice that, since the weight vectors are required to be non-negative, $\bar{w}_z = w^1_z = w^2_z = 0$ for every $z \in V \setminus \{u,v\}$. Thus, we can assume without loss of generality that $w^1_u = w^2_v = 1+\epsilon$ and $w^1_v = w^2_u = 1-\epsilon$ for some $\epsilon > 0$. However, the fully half-integral solution is now not an optimal LP solution on the weights w^1 and w^2. Indeed, on the weight function w^1, the feasible solution $V \setminus \{u\}$ has objective value $1-\epsilon$, whereas the fully half-integral solution has weight 1. Similarly, on the weight function w^2, the feasible solution $V \setminus v$ has objective value $1-\epsilon$. By Lemma 3.4.1, $w^1, w^2 \notin Q^W$, leading to a contradiction. The weight function $\bar{w}$ thus cannot be written as a non-trivial convex combination of two other points in the polytope Q^W and is therefore an extreme point (or a vertex) of Q^W. □

We end this section by showing that this normalization of the weight space allows us to get a convenient lower bound on $w(\mathsf{OPT}(\mathcal{G}))$.

Lemma 3.4.3. *Let $\mathcal{G} = (V,E)$ be a graph. For any $w \in Q^W$, $w(\mathsf{OPT}(\mathcal{G})) \geq 1$.*

Proof:
Since $w \in Q^W$, we know that the fully half-integral solution is an optimal linear programming solution, showing $1 = w(V)/2 \leq w(\mathsf{OPT}(\mathcal{G}))$, by feasibility of $\mathsf{OPT}(\mathcal{G})$. □

3.5 Analysis of the algorithm

This section is devoted to the analysis of the approximation ratio of the algorithm. We assume that we are given as input a pair $(\mathcal{G}, S)$ consisting of a weighted graph and an odd cycle transversal $S \subset V$. By Corollary 3.3.2, we may assume that the weight function satisfies $w \in Q^W$. By the previous section, there are dual edge weights $y_e \geq 0$ such that $w_v = y(\delta(v))$ for every $v \in V$ and which satisfy $\sum_{e \in E} y_e = 1$.

The algorithm is now very simple. First, take the vertices in $S \subset V$ to the cover and remove them from the graph. Then solve the integral linear program

$P(\mathcal{G} \setminus S)$ and take the vertices having LP value one to the cover. The *approximation ratio*, given a weight function $w \in Q^W$, is thus defined as

$$R(w) := \frac{w(S) + w(\mathsf{OPT}(\mathcal{G} \setminus S))}{w(\mathsf{OPT}(\mathcal{G}))}. \tag{3.5.1}$$

For simplicity of notation, we omit the dependence on w of $\mathsf{OPT}(\mathcal{G})$ and $\mathsf{OPT}(\mathcal{G} \setminus S)$. As a reminder, the bipartite graph $\mathcal{G} \setminus S$ is denoted by $\mathcal{G}' = (V', E')$. The vertex contracted graph $\mathcal{G}/S$ is denoted by $\tilde{\mathcal{G}} = (\tilde{V}, \tilde{E})$ and the contracted node is denoted by v_S.

3.5.1 Stable set to bipartite

We assume in this section that S is a stable set. We will then generalize the results obtained here in a natural way to the most general setting of an arbitrary set S. We now state our main theorem of this section.

Theorem 3.5.1. *Let $(\mathcal{G}, S)$ be the input to the rounding algorithm, with S being a stable set. For any $w \in Q^W$, the approximation ratio satisfies*

$$R(w) \leq 1 + \frac{1}{\rho}$$

where $2\rho - 1$ is the odd girth of the contracted graph $\tilde{\mathcal{G}}$ and satisfies $\rho \in [2, \infty]$. Moreover, this bound is tight and is attained for a class of weight functions $\mathcal{W} \subset Q^W$.

Remark 3.5.2. We define the odd girth of a bipartite graph as being ∞.

Definition 3.5.3. Let $(\mathcal{G}, S)$ be a pair consisting of a graph with an odd cycle transversal S. For a feasible vertex cover $U \subset V \setminus S$ of the bipartite graph $\mathcal{G}' = \mathcal{G} \setminus S$, we define

$$E_U := \big\{(u, v) \in E \;\big|\; u \in U,\ v \in U \text{ or } \ u \in U,\ v \in S\big\}.$$

In words, these are the edges with either both endpoints in the cover U, or with one endpoint in U and one in S.

Definition 3.5.4. Let $(\mathcal{G}, S)$ be a pair consisting of a graph with an odd cycle transversal S. Feasible vertex covers $U_1, \ldots, U_k$ of the bipartite graph $\mathcal{G}' = \mathcal{G} \setminus S$ are defined to be *pairwise edge-separate* if the edge sets $\{E_{U_1}, \ldots, E_{U_k}\}$ are pairwise disjoint.

Remark 3.5.5. We will often say that covers are pairwise edge-separate for the pair $(\mathcal{G}, S)$. It is however worth emphasizing that these covers are defined on the bipartite graph $\mathcal{G}' = \mathcal{G} \setminus S$.

This definition turns out to be the key concept for us in order to prove improved bounds on the approximation ratio of the algorithm, as shown by the next lemma.

Lemma 3.5.6. *Let $(\mathcal{G}, S)$ be the input to the rounding algorithm, with S being a stable set. If there exists k pairwise edge-separate feasible vertex covers of the bipartite graph $\mathcal{G}' = \mathcal{G} \setminus S$, then, for every $w \in Q^W$, the approximation ratio of the algorithm satisfies*

$$R(w) \leq 1 + \frac{1}{k}.$$

Proof:
Let $w \in Q^W$ and let $y \in \mathbb{R}_+^E$ be the corresponding dual solution satisfying $w_v = y(\delta(v))$ and $y(E) = 1$. We denote by $\{U_1, \ldots, U_k\}$ the pairwise edge-separate covers of $\mathcal{G}' = (V', E')$. We can now write down the weights of S and every feasible cover U_i with the help of the dual variables:

$$\begin{aligned} w(S) &= \sum_{v \in S} w_v = \sum_{v \in S} y(\delta(v)) = y(\delta(S)) \\ w(U_i) &= \sum_{v \in U_i} w_v = \sum_{v \in U_i} y(\delta(v)) = y(E') + y(E_{U_i}) \qquad \forall i \in [k] \end{aligned}$$

The first equality holds because S is a stable set and thus only has edges crossing the set. The second equality holds because every U_i counts the dual value y_e of every $e \in E'$ at least once, by feasibility of the cover, and thus giving a contribution of $y(E')$. The edges in E_{U_i} then give an additional contribution of $y(E_{U_i})$.

By Lemma 3.4.3, the approximation ratio satisfies:

$$\begin{aligned} R(w) = \frac{w(S) + w(\mathsf{OPT}(\mathcal{G} \setminus S))}{w(\mathsf{OPT}(\mathcal{G}))} &\leq w(S) + w(\mathsf{OPT}(\mathcal{G}')) \leq w(S) + \min_{i \in [k]} w(U_i) \\ = y(\delta(S)) + y(E') + \min_{i \in [k]} y(E_{U_i}) &= 1 + \min_{i \in [k]} y(E_{U_i}) \leq 1 + \frac{1}{k}. \end{aligned}$$

The last equality follows from the fact that $E = E' \cup \delta(S)$ and $y(E) = 1$. The last inequality follows from the fact that the edge sets $\{E_{U_i}\}_{i \in [k]}$ are pairwise disjoint and have a dual sum of at most one, since the total sum of the edges of the graph is $y(E) = 1$. This minimum can thus be upper bounded by $1/k$. □

In order to prove the upper bound in Theorem 3.5.1, we thus need to construct ρ pairwise edge-separate covers of $\mathcal{G}' = \mathcal{G} \setminus S$. The key for being able to do that is to analyze the structure of the contracted graph $\tilde{\mathcal{G}} = \mathcal{G}/S$, where S is contracted into a single node v_S.

Lemma 3.5.7. *Let $(\mathcal{G}, S)$ be a graph with an odd cycle transversal S. If the contracted graph $\tilde{\mathcal{G}}$ contains an odd cycle, then there exists ρ edge-separate feasible covers for the pair $(\tilde{\mathcal{G}}, v_S)$, where $2\rho - 1$ is the odd girth of $\tilde{\mathcal{G}}$.*

Proof:
We now dive deeper into the structure of the bipartite graph $\mathcal{G} \setminus S = \tilde{\mathcal{G}} \setminus v_S$. By assumption, this graph admits a bipartition $A \cup B$ of the vertices. Let us assume that it has k connected components $A_1 \cup B_1, \ldots, A_k \cup B_k$, all of which are bipartite as well, where $A = \bigcup_i A_i$ and $B = \bigcup_i B_i$. We now fix an arbitrary such component $A_j \cup B_j$.

- If v_S has an incident edge to both A_j and B_j, then this component contains (if including v_S) an odd cycle of $\tilde{\mathcal{G}}$. This holds since any path between a node in A_j and a node in B_j has odd length.
- If v_S has incident edges with only one side, we assume without loss of generality that this side is A_j. One could simply switch both sides in the other case while still keeping a valid bipartition of the graph $\tilde{\mathcal{G}} \setminus v_S$.
- If v_S does not have incident edges with either of the two sides, then $A_j \cup B_j$ is a connected component of $\tilde{\mathcal{G}}$. We call such components *dummy* components and denote by $A_d \cup B_d$ the bipartite graph formed by taking the union of all the dummy components.

We denote $N_A(v_S) = N(v_S) \cap A$ and $N_B(v_S) = N(v_S) \cap B$. We now split the graph into layers, where each layer corresponds to the nodes at the same shortest path distance from $N_A(v_S)$. More precisely, we define

$$\mathcal{L}_i := \Big\{ v \in A \cup B \mid d(N_A(v_S), v) = i \Big\} \quad \text{for } i \in \{0, \ldots, q\} \tag{3.5.2}$$

where $d(N_A(v_S), v)$ represents the unweighted shortest path distance between v and a vertex in $N_A(v_S)$. The parameter q is defined to be the maximal finite distance from $N_A(v_S)$ in the graph $\tilde{\mathcal{G}}$. An important observation is the fact that these layers are alternatingly included in one side of the bipartition, see Figure 3.1 for an illustration of the construction.

If the graph $\tilde{\mathcal{G}}$ is not connected, note that $d(N_A(v_S), v) = \infty$ for the vertices v lying in dummy components. In order to add the dummy components to the layers and keep alternation between the two sides of the bipartition, we define the last two layers to either be $\{\mathcal{L}_{q+1} := A_d,\ \mathcal{L}_{q+2} := B_d\}$ or $\{\mathcal{L}_{q+1} := B_d,\ \mathcal{L}_{q+2} := A_d\}$, depending on which side of the bipartition the last connected layer $\mathcal{L}_q$ lies. We now have that $\mathcal{L}_i \subset A$ if i is even, and $\mathcal{L}_i \subset B$ if i is odd. In fact,

$$A = \bigcup_{i=0}^{\lfloor l/2 \rfloor} \mathcal{L}_{2i} \quad \text{and} \quad B = \bigcup_{i=1}^{\lceil l/2 \rceil} \mathcal{L}_{2i-1},$$

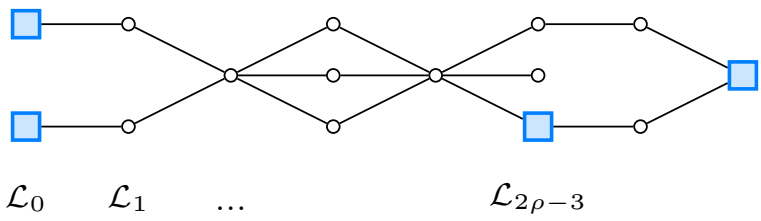


Figure 3.1: The layers of a bipartite graph $\tilde{\mathcal{G}} \setminus v_S = (A \cup B, E')$ with $\rho = 4$. The blue square vertices correspond to $N(v_S)$, where the two left ones are $\mathcal{L}_0 = N_A(v_S)$ and the two right ones are $N_B(v_S)$.

where the parameter $l \in \mathbb{N}$ represents the index of the last layer: if $\tilde{\mathcal{G}}$ is connected, then $l = q$, otherwise $l = q + 2$. Notice also that $\mathcal{L}_0 = N_A(v_S)$. However, $N_B(v_S)$ may now have several different vertices in different layers, see Figure 3.1.

Let $C \subset V$ be an arbitrary odd cycle of $\tilde{\mathcal{G}}$. Notice that this cycle contains v_S, a vertex from $N_A(v_S)$ and a vertex from $N_B(v_S)$, since $\tilde{\mathcal{G}} \setminus v_S$ is bipartite and therefore does not contain an odd cycle. Any odd cycle C in $\tilde{\mathcal{G}}$ thus corresponds to an odd path between a vertex in $N_A(v_S) = \mathcal{L}_0$ and a vertex in $N_B(v_S)$. By the assumption that the shortest odd cycle length of $\tilde{\mathcal{G}}$ is $2\rho - 1$, the first layer having a non-empty intersection with $N_B(v_S)$ is $\mathcal{L}_{2\rho-3}$. A shortest odd cycle of length $2\rho - 1$ therefore corresponds to an odd path of length $2\rho - 3$ between $\mathcal{L}_0$ and a vertex in $\mathcal{L}_{2\rho-3} \cap N_B(v_S)$, see Figure 3.1 for an illustration. We now define edges connecting two consecutive layers $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$ as follows:

$$E[\mathcal{L}_i, \mathcal{L}_{i+1}] := \{(u, v) \in E' \mid u \in \mathcal{L}_i, v \in \mathcal{L}_{i+1}\} \qquad \forall i \in \{0, \dots, l-1\}.$$

We also denote by

$$\delta_A(v_S) = \{(v_S, u) \in \tilde{E} \mid u \in A\}, \qquad \delta_B(v_S) = \{(v_S, u) \in \tilde{E} \mid u \in B\}$$

the incident edges to v_S respectively connecting to A and B.

We are now ready to construct our desired ρ pairwise edge-separate covers of $\tilde{\mathcal{G}} \setminus v_S$, that we denote by $U_1, \dots, U_\rho$ and illustrated in Figure 3.2. Firstly, notice that taking one side of the bipartition is a feasible vertex cover. We thus define $U_\rho = A$ and $U_{\rho-1} = B$. Observe that we then have $E_{U_\rho} = \delta_A(v_S)$ and $E_{U_{\rho-1}} = \delta_B(v_S)$.

We now construct $\rho - 2$ additional covers with the help of the layers. If $\rho \neq 2$, fix a $j \in [\rho - 2]$, and start the cover U_j by taking the two consecutive layers $\mathcal{L}_{2j-1}$ and $\mathcal{L}_{2j}$. Complete this cover by taking remaining layers alternatingly (hence always skipping one) until covering every edge of the graph. Notice that this cover has an empty intersection with $N(v_S)$. We then have that

$$E_{U_\rho} = \delta_A(v_S), \qquad E_{U_{\rho-1}} = \delta_B(v_S), \qquad E_{U_j} = E[\mathcal{L}_{2j-1}, \mathcal{L}_{2j}] \quad \forall j \in [\rho - 2],$$

which are all pairwise disjoint edge sets, finishing the proof. □

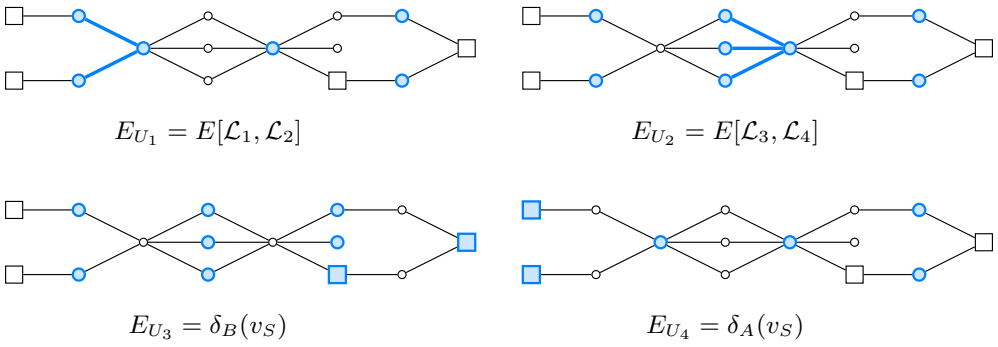


Figure 3.2: The ρ feasible covers of $\mathcal{G}'$ constructed in the proof of Lemma 3.5.7.

We now have all the tools to prove the upper bound of Theorem 3.5.1.

Proof:

Asume first that $\rho < \infty$, meaning that $\tilde{\mathcal{G}}$ contains an odd cycle. By Lemma 3.5.7, there exists ρ pairwise edge-separate covers for the pair $(\tilde{\mathcal{G}}, v_S)$. These covers are then still edge-separate for the pair $(\mathcal{G}, S)$, since the bipartite graph is the same in both cases, i.e. $\mathcal{G}' = \tilde{\mathcal{G}} \setminus v_S = \mathcal{G} \setminus S$. This finishes the proof by Lemma 3.5.6.

If $\rho = \infty$, then $\tilde{\mathcal{G}}$ is bipartite, with a bipartition $\tilde{A} \cup \tilde{B}$. Assume without loss of generality that $v^S \in \tilde{A}$. Note that $\tilde{E} = E' \cup \delta(v^S)$ and thus $1 = y(\tilde{E}) = y(E') + y(\delta(v^S))$. Any feasible cover of $\mathcal{G}' = \mathcal{G} \setminus S$ needs to count the dual value of every edge in E' at least once. Taking the cover $\tilde{A} \setminus v^S$ counts every edge in E' exactly once, showing that $w(\mathsf{OPT}(\mathcal{G} \setminus S)) = y(E')$. Hence, $R(w) \leq w(S) + w(\mathsf{OPT}(\mathcal{G} \setminus S)) = y(\delta(v_S)) + y(E') = 1$. □

We now show that this bound is tight and is attained for a class of weight functions $w \in \mathcal{W}$ for any such graph $\mathcal{G}$ and stable set S. For the case where $\rho = \infty$, it is clear that the approximation ratio always satisfies $R(w) \geq 1$, showing that the bound in Theorem 3.5.1 is tight for any $w \in Q^W$.

We thus assume that $\rho < \infty$. Let $\mathcal{C}$ be all the shortest odd cycles (of length $2\rho - 1$) of the graph $\tilde{\mathcal{G}}$, each of which is containing v_S. For every such cycle $C \in \mathcal{C}$, we define the following dual function on the edges $y^C : \tilde{E} \to \mathbb{R}_+$: set both dual edges incident to v_S to $1/\rho$ and then alternatingly set the dual edges to 0 and $1/\rho$ along the odd cycle. For any edge outside of C, set its dual value to 0. Such a solution clearly satisfies $y^C(\tilde{E}) = 1$. We now take the convex hull of all these functions:

$$\mathcal{Y} := \left\{ y : \tilde{E} \to \mathbb{R}_+ \mid y = \sum_{C \in \mathcal{C}} \lambda^C y^C, \quad \sum_{C \in \mathcal{C}} \lambda^C = 1, \quad \lambda^C \geq 0 \quad \forall C \in \mathcal{C} \right\}.$$

Because of the one-to-one correspondence between the edge sets $\tilde{E}$ and E, due to the fact that S is a stable set, we can naturally define a weight function on the original vertex set once we fix a $y \in \mathcal{Y}$ by setting $w_v := y(\delta(v))$ for every $v \in V$.

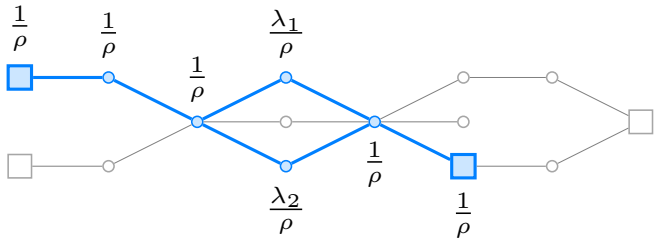


Figure 3.3: An example of a weight function $w \in \mathcal{W}$ obtained by a convex combination of two basic weight functions of shortest odd cycles.

We define the space of all such weight functions as

$$\mathcal{W} := \{w : V \to \mathbb{R}_+ \mid w_v = y(\delta(v)) \quad \forall v \in V, \quad y \in \mathcal{Y}\}.$$

Theorem 3.5.8. *For any weight function $w \in \mathcal{W}$, the approximation ratio satisfies*

$$R(w) = 1 + \frac{1}{\rho}$$

where $2\rho - 1$ is the odd girth of $\tilde{\mathcal{G}}$ and satisfies $\rho \in [2, \infty)$.

Proof:
Let $\mathcal{C}$ be the set of all the shortest odd cycles (of length $2\rho - 1$) of the graph $\tilde{\mathcal{G}}$ and let $w \in \mathcal{W}$ with the corresponding $y = \sum_{C \in \mathcal{C}} \lambda^C y^C$. Notice that, for any subset of vertices $U \subset V'$ of the bipartite graph $\mathcal{G}'$, we can count its weight as

$$\begin{aligned} w(U) &= \sum_{v \in U} w_v = \sum_{v \in U} y(\delta(v)) = \sum_{v \in U} \sum_{C \in \mathcal{C}} \lambda^C y^C(\delta(v)) \\ &= \sum_{v \in U} \sum_{C \in \mathcal{C}} \frac{\lambda^C}{\rho} 1_{\{v \in C\}} = \frac{1}{\rho} \sum_{C \in \mathcal{C}} \lambda^C \sum_{v \in U} 1_{\{v \in C\}} = \frac{1}{\rho} \sum_{C \in \mathcal{C}} \lambda^C |U \cap C|. \end{aligned} \tag{3.5.3}$$

The end of the proof now heavily uses the decomposition of $\tilde{\mathcal{G}}$ into layers described in (3.5.2). Notice that every odd cycle $C \in \mathcal{C}$ intersects each layer $\mathcal{L}_i$ for $i \in \{0, \dots, 2\rho - 3\}$ exactly once. Therefore, by (3.5.3), $w(\mathcal{L}_i) = 1/\rho$ for every $i \in \{0, \dots, 2\rho - 3\}$. We now claim that

$$w(\mathsf{OPT}(\mathcal{G})) = 1, \quad w(\mathsf{OPT}(\mathcal{G}')) = \frac{\rho - 1}{\rho} \quad \text{and} \quad w(S) = \frac{2}{\rho}.$$

The fact that $w(\mathsf{OPT}(\mathcal{G})) \geq 1$ follows from Lemma 3.4.3. For the reverse inequality, notice that it is possible to take a feasible cover by taking exactly ρ layers in addition to the zero weight vertices, for instance $\mathcal{L}_0 \cup \mathcal{L}_2 \cup \mathcal{L}_3 \cup \mathcal{L}_5 \cdots \cup \mathcal{L}_{2\rho-3}$, showing $w(\mathsf{OPT}(\mathcal{G})) \leq 1$.

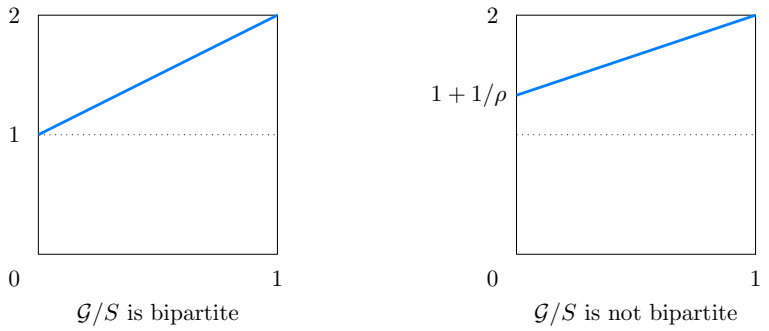


Figure 3.4: The plot of the approximation ratio $R(w)$ with respect to the parameter $\alpha \in [0,1]$.

Observe now that $w(\mathsf{OPT}(\mathcal{G}')) = w(\mathsf{OPT}(\mathcal{G} \setminus S)) = w(\mathsf{OPT}(\tilde{\mathcal{G}} \setminus v_S))$. After removal of v_S, every cycle $C \in \mathcal{C}$ becomes a path of length $2\rho - 3$ (and thus consisting of $2\rho-2$ vertices), with one vertex in each layer $\mathcal{L}_i$ for $i \in \{0, \dots, 2\rho-3\}$. By feasibility, $\mathsf{OPT}(\mathcal{G}')$ has to contain at least $\rho - 1$ vertices for every such path. Using (3.5.3), we infer $w(\mathsf{OPT}(\mathcal{G}')) \geq (\rho-1)/\rho$. For the reverse inequality, taking $\rho - 1$ layers alternatively, such as $\mathcal{L}_0 \cup \mathcal{L}_2 \cup \mathcal{L}_4 \cdots \cup \mathcal{L}_{2\rho-4}$, as well as the zero weight vertices, builds a feasible cover of weight exactly $(\rho - 1)/\rho$.

Finally, notice that $w(S) = w(v_S) = 2/\rho$ because every $C \in \mathcal{C}$ contains v_S. By combining the three equalities, we get the desired result

$$R(w) = \frac{w(S) + w(\mathsf{OPT}(\mathcal{G}'))}{w(\mathsf{OPT}(\mathcal{G}))} = 1 + \frac{1}{\rho}.$$

□

3.5.2 Arbitrary set to bipartite

We now consider the setting where S is now an arbitrary set. Our guarantee on the approximation ratio will now also depend on the total sum of the dual variables on the edges inside of the set S. We denote this sum by

$$\alpha := y(E[S]) \in [0,1].$$

Theorem 3.5.9. *For any $w \in Q^W$, the approximation ratio satisfies*

$$R(w) \leq \left(1 + \frac{1}{\rho}\right)(1-\alpha) + 2\alpha \qquad \text{with } \alpha \in [0,1] \text{ and } \rho \in [2,\infty]$$

where $2\rho - 1$ denotes the odd girth of the contracted graph $\tilde{\mathcal{G}}$. Moreover, these bounds are tight and are attained for any $\alpha \in [0,1]$ and any $\rho \in [2,\infty]$.

Proof:

Upper bound for $\rho < \infty$. We consider first the case where $\rho < \infty$. The proof essentially follows the same arguments as the one of Lemma 3.5.6 with the α parameter incorporated, and we thus only highlight the main differences. We decompose the weight of the set S with respect to the dual variables. The edges in $E[S]$ are counted twice, whereas the edges in $\delta(S)$ are counted once:

$$w(S) = 2\alpha + y(\delta(S)). \tag{3.5.4}$$

Consider the contracted graph $\tilde{\mathcal{G}} = \mathcal{G}/S = (\tilde{V}, \tilde{E})$ and denote by v^S the contracted node. The edge set of this graph is now $\tilde{E} = \delta(S) \cup E'$, since the edges in $E[S]$ have been collapsed. By Lemma 3.5.7, we can construct ρ edge-separate covers $U_1, \ldots, U_\rho \subset \tilde{V} \setminus v^S$ for the pair $(\tilde{\mathcal{G}}, v_S)$. These covers are still edge-separate for the pair $(\mathcal{G}, S)$, implying

$$w(\mathsf{OPT}(\mathcal{G} \setminus S)) \leq \min_{i\in[\rho]} w(U_i) = y(E') + \min_{i\in[\rho]} y(E_{U_i}) \leq y(E') + \frac{1-\alpha}{\rho}. \tag{3.5.5}$$

The first inequality holds since every U_i is a feasible cover of $\mathcal{G} \setminus S$. The second equality holds by counting the weight of a cover U_i in terms of the dual edges. The last inequality holds because the edge sets $\{E_{U_i}\}_{i\in[\rho]}$ are pairwise disjoint, and their total dual sum is at most $1 - \alpha$. Combining Lemma 3.4.3, (3.5.4) and (3.5.5),

$$R(w) \leq 2\alpha + y(\delta(S)) + y(E') + \frac{1-\alpha}{\rho} = 1 + \alpha + \frac{1-\alpha}{\rho} = \left(1 + \frac{1}{\rho}\right)(1-\alpha) + 2\alpha.$$

Upper bound for $\rho = \infty$. The only change with respect to the previous proof is the bound on $w(OPT(\mathcal{G} \setminus S))$ in (3.5.5). We denote the contracted graph by $\tilde{\mathcal{G}} = \mathcal{G}/S$ and by v^S the contracted vertex. Suppose $\tilde{\mathcal{G}}$ admits the bipartition $(\tilde{A} \cup \tilde{B}, \tilde{E})$ and assume without loss of generality that $v^S \in \tilde{A}$. Note that $\tilde{E} = E' \cup \delta(v^S)$.

Any feasible cover of $\mathcal{G} \setminus S$ needs to count the dual value of every edge in E' at least once. Taking the cover $\tilde{A} \setminus v^S$ counts every edge in E' exactly once, showing that $w(OPT(\mathcal{G} \setminus S)) = y(E')$. Hence, using $y(E) = \alpha + y(\delta(S)) + y(E') = 1$, we get

$$R(w) \leq 2\alpha + y(\delta(S)) + y(E') = 1 + \alpha.$$

Tight instance for $\rho < \infty$. An example of a tight instance can be constructed as follows. We first construct $\mathcal{G}/S$: take an odd cycle of length $2\rho - 1$ with a distinguished node v^S and assign dual value $(1-\alpha)/\rho$ to both edges incident to it. Alternatively assign dual values 0 and $(1-\alpha)/\rho$ along the odd cycle for the remaining edges. In order to construct $\mathcal{G}$, replace v^S by a triangle S with dual edges set to $\alpha, 0$ and 0, where the two previous incident edges to v^S are adjacent to the endpoints of the edge with value α. Note that we replace it with a triangle

instead of a single edge in order to avoid $\mathcal{G}$ becoming bipartite. Similarly to the proof of Theorem 3.5.8, one can check that

$$w(S) = 2\alpha + \frac{2\,(1-\alpha)}{\rho}; \quad w(OPT(\mathcal{G} \setminus S)) = \frac{(1-\alpha)(\rho-1)}{\rho}; \quad w(OPT(\mathcal{G})) = 1.$$

Therefore,

$$R(w) = 2\alpha + \frac{2\,(1-\alpha)}{\rho} + \frac{(1-\alpha)(\rho-1)}{\rho} = \left(1 + \frac{1}{\rho}\right)(1-\alpha) + 2\alpha.$$

Tight instance for $\rho = \infty$. Let $\mathcal{G}$ be an arbitrary odd cycle. Consider an arbitrary edge $(u, v) \in E$ and assign it dual value α. The set S is defined to be $S = \{u, v\}$. Assign dual value zero to the edge $(u, w) \in E$, where w is the second neighbour of u in the cycle. For the remaining edges, arbitrarily assign dual values, while ensuring that they sum up to $1 - \alpha$. The fact that one edge is equal to zero is necessary in order to get the exact formula $w(OPT(\mathcal{G})) = 1$, a feasible cover showing $w(OPT(\mathcal{G})) \leq 1$ being the following: take both endpoints of the edge (u, w) and take remaining vertices alternatively (hence always skipping one) along the odd cycle. All the edges are counted once, except for (u, w), which is counted twice but has value zero. Moreover, $w(S) = 2\alpha + y(\delta(S))$ and $w(OPT(\mathcal{G} \setminus S)) = y(E')$, where E' is the edge set of the bipartite graph $\mathcal{G} \setminus S$. Therefore,

$$R(w) = 2\alpha + y(\delta(S)) + y(E') = 1 + \alpha.$$

□

3.6 Algorithmic applications

We focus in this section on efficient ways to find odd cycle transversals with a low value for the α parameter. In fact, once such a set S is found, there can also be freedom in the choice of the dual solution in order to optimize the α parameter. This motivates the following definition.

Definition 3.6.1. Let $(\mathcal{G}, S, y, w)$ be a graph with an odd cycle transversal $S \subset V$, weights $w \in Q^W$ and a dual solution $y \in \mathbb{R}_+^E$. A tuple $(\mathcal{G}', S', y', w')$ is *approximation preserving* if

$$w(S) + w(\mathsf{OPT}(\mathcal{G} \setminus S)) \leq w'(S') + w'(\mathsf{OPT}(\mathcal{G}' \setminus S')).$$

Moreover, we say that $\alpha \in [0, 1]$ is *valid* for the pair $(\mathcal{G}, S)$ if there exists an approximation preserving $(\mathcal{G}', S', y', w')$ such that $\alpha = y'(E[S'])$.

Finding a valid $\alpha \in [0,1]$ would directly allow us to use it in the bound of Theorem 3.5.9, where the ρ parameter would correspond to the one of the approximation preserving graph. We present here an application if a coloring of a graph can be found efficiently.

Theorem 3.6.2. *Let $\mathcal{G} = (V, E)$ be a graph with weights $w \in Q^W$ that can be k-colored in polynomial time for $k \geq 4$. There exists an efficiently findable set $S \subset V$ bipartizing the graph and a valid α such that $\alpha \leq 1 - 4/k$, leading to an approximation ratio of*

$$R(w) \leq 2 - \frac{4}{k}\left(1 - \frac{1}{\rho}\right).$$

Proof:
Let us denote by $V_1, \ldots, V_k$ the k independent sets defining the color classes of the graph $\mathcal{G}$. We assume without loss of generality that they are ordered by weight $w(V_1) \leq w(V_2) \cdots \leq w(V_k)$. Since $w(V) = 2$, the two color classes with the largest weights satisfy $w(V_{k-1}) + w(V_k) \geq 4/k$. We define the bipartizing set to be the remaining color classes: $S := V_1 \cup \cdots \cup V_{k-2}$. We denote by $y \in \mathbb{R}_+^E$ the dual solution satisfying complementary slackness and $y(E) = 1$.

We now define an approximation preserving $(\mathcal{G}', S', y', w')$ in the following way. Let $\mathcal{G}' = K_k$ be the complete graph on k vertices, denoted by $\{v_1, \ldots, v_k\}$. The weights are defined to be

$$w'(v_i) := w(V_i) \quad \text{and} \quad y'(v_i, v_j) := y(E[V_i, V_j])$$

for every $i, j \in [k]$. These clearly satisfy the complementary slackness condition $y'(\delta(v_i)) = w'(v_i)$ for every $i \in [k]$. The bipartizing set is defined to be $S' := \{v_1, \ldots, v_{k-2}\}$. This tuple is approximation preserving since $w(S) = w'(S')$ and $w(\mathsf{OPT}(\mathcal{G} \setminus S)) \leq w'(\mathsf{OPT}(\mathcal{G}' \setminus S'))$. In order to prove the theorem, we still need to tweak the dual solution y' to ensure $\alpha := y'(E[S']) \leq 1 - 4/k$. Observe that $w'(v_{k-1}) + w'(v_k) \geq 4/k$.

1. If $y'(v_{k-1}, v_k) = 0$, then the result follows since in that case $y'(\delta(S')) \geq 4/k$ and thus $y'(E[S']) \leq 1 - 4/k$.

2. If $y'(E[S']) = 0$, then the result trivially follows as well.

Suppose thus that $y'(E[S']) > 0$ and $y'(v_{k-1}, v_k) > 0$. Pick an arbitrary edge $(v_i, v_j) \in E[S']$ satisfying $y'(v_i, v_j) > 0$ and consider the 4-cycle (v_i, v_j, v_{k-1}, v_k). Notice that alternatively increasing and decreasing the dual values on the edges of this cycle by a small amount $\epsilon > 0$ gives another feasible dual solution satisfying the complementary slackness condition. More formally, we set $\epsilon := \min\{y'(v_i, v_j), y'(v_{k-1}, v_k)\}$, decrease $y'(v_i, v_j)$ and $y'(v_{k-1}, v_k)$ by ϵ, while increasing $y'(v_j, v_{k-1})$ and $y'(v_k, v_i)$ by the same amount. Observe that this leads to

either (v_i, v_j) or (v_{k-1}, v_k) dropping to dual value zero. We can repeat this procedure until either $y'(E[S']) = 0$ or $y'(v_{k-1}, v_k) = 0$, finishing the proof of the theorem. □

We now claim that this result is optimal in the following sense. Consider an n-vertex graph. It is known that the integrality gap of the standard linear programming relaxation for vertex cover is upper bounded by $2 - 2/n$, a bound which is attained on the complete graph. This implies that any approximation algorithm lower bounding $w(\mathsf{OPT})$ by comparing it to the optimal LP solution, as we do in Lemma 3.4.3, cannot do better than $2-2/n$ in the worst case. Setting $\rho = 2$ in Theorem 3.6.2, which corresponds to the worst case since $\rho \in [2, \infty]$, recovers this bound and a result of Hochbaum in [Hoc83].

3.7 Integrality gap and fractional chromatic number

We focus in this section on proving tight bounds for the integrality gap of 3-colorable graphs. A key result that we use in this section is given by Singh in [Sin19], which relates the integrality gap with the fractional chromatic number of a graph. The latter is denoted as $\chi^f(\mathcal{G})$ and is defined as the optimal solution of the following primal-dual linear programming pair. We denote by $\mathcal{I} \subset 2^V$ the set of all independent sets of the graph $\mathcal{G}$. Solving these linear programs is however NP-hard because of the possible exponential number of independent sets.

$$\begin{aligned} \min \sum_{I \in \mathcal{I}} y_I & \qquad & \max \sum_{v \in V} z_v \\ \sum_{I \in \mathcal{I}, v \in I} y_I \geq 1 \quad \forall v \in V & & \sum_{v \in I} z_v \leq 1 \quad \forall I \in \mathcal{I} \\ y_I \geq 0 \quad \forall I \in \mathcal{I} & & z_v \geq 0 \quad \forall v \in V \end{aligned}$$

Note that $\chi^f(\mathcal{G}) = 2$ if and only if $\mathcal{G}$ is bipartite.

Theorem. 3.7.1 (Singh, [Sin19]). *Let $\mathcal{G} = (V, E)$ be a graph. The integrality gap of the vertex cover linear programming relaxation $P(\mathcal{G})$ satisfies*

$$IG(\mathcal{G}) = 2 - \frac{2}{\chi^f(\mathcal{G})}.$$

We first focus on graphs with the existence of a single vertex whose removal produces a bipartite graph. The following theorem generalizes the result given for the cycle graph in [ABL02, SU11] and turns out to be the same formula as for series-parallel graphs [GX16].

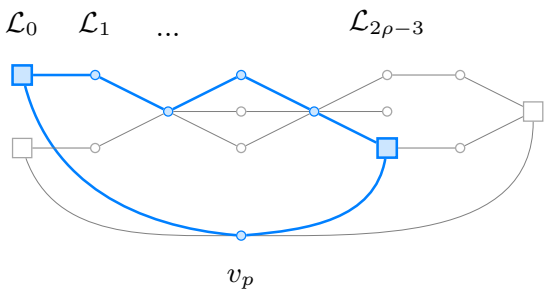


Figure 3.5: An optimal dual solution constructed in the proof of Theorem 3.7.2. Each node on a shortest odd cycle is assigned a fractional value of $1/(\rho-1)$.

Theorem 3.7.2. *Let $\mathcal{G} = (V, E)$ be a non-bipartite graph and $v_p \in V$ such that $\mathcal{G} \setminus v_p = (A \cup B, E')$ is bipartite. Then,*

$$\chi^f(\mathcal{G}) = 2 + \frac{1}{\rho - 1},$$

where $2\rho - 1$ is the odd girth of $\mathcal{G}$.

Proof of Theorem 3.7.2:
We prove this theorem by constructing feasible primal and dual solutions of objective value $2 + 1/(\rho - 1)$. By weak duality, these two solutions are then optimal for their respective linear programs, hence proving the theorem.

We first construct the dual solution. Let $\mathcal{C}$ be the set of all the shortest odd cycles of $\mathcal{G}$. For any such cycle $C \in \mathcal{C}$, define the dual solution $z^C \in \mathbb{R}^V$ by

$$z_v^C = \begin{cases} 1/(\rho-1) & \text{if } v \in C \\ 0 & \text{if } v \in V \setminus C \end{cases}$$

This solution is feasible since any independent set in an odd cycle of length $2\rho-1$ has size at most $\rho - 1$. Indeed, fix an independent set $I \in \mathcal{I}$, then:

$$\sum_{v \in I} z_v^C = \sum_{v \in I \cap C} \frac{1}{\rho - 1} = \frac{|I \cap C|}{\rho - 1} \leq 1.$$

Moreover, the objective value of this solution is:

$$\sum_{v \in V} z_v^C = \sum_{v \in C} \frac{1}{\rho - 1} = \frac{2\rho - 1}{\rho - 1} = 2 + \frac{1}{\rho - 1}.$$

Let us now construct the primal solution. We will do so by constructing $2\rho - 1$ independent sets $I_k \in \mathcal{I}$ and assigning to each of them a fractional value of $y(I_k) = 1/(\rho - 1)$. All the other independent sets are assigned value zero. We split the bipartite graph $\mathcal{G} \setminus v_p$ into the layers

$$\mathcal{L}_i := \{v \in A \cup B \mid d(N_A(v_p), v) = i\} \quad \text{for } i \in \{0, \ldots, l\}.$$

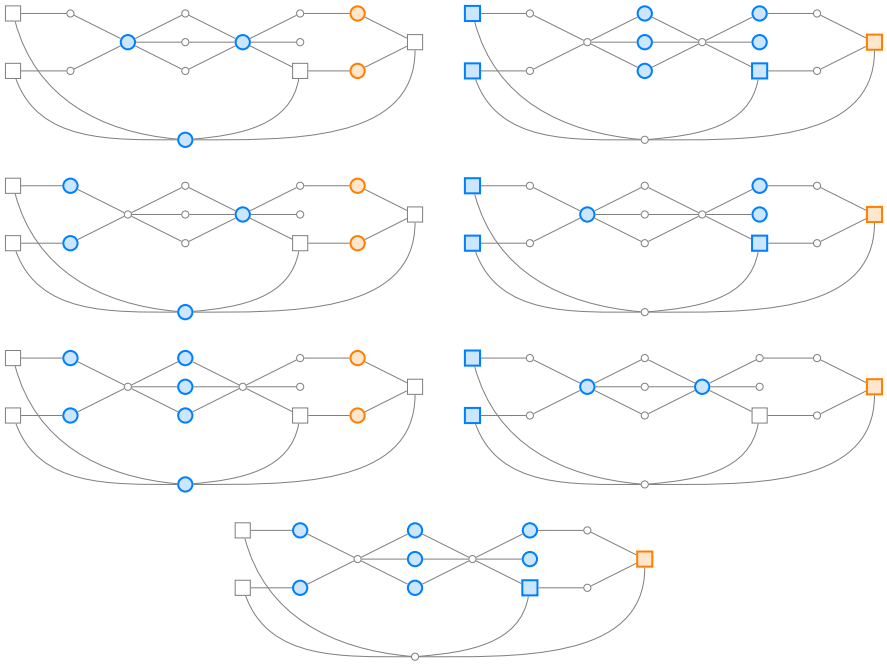


Figure 3.6: The $2\rho - 1$ independent sets I_k constructed in the optimal primal solution. The blue nodes correspond to $\{U_k \mid k \in [2\rho - 1]\}$, whereas the orange nodes correspond to $\{R_1, R_2\}$.

as explained in (3.5.2). As a reminder, any shortest odd cycle corresponds to a path between $\mathcal{L}_0 = N_A(v_p)$ and $\mathcal{L}_{2\rho-3} \cap N_B(v_p)$. The original vertex set V is thus decomposed into $\{v_p\} \cup \mathcal{L}_0 \cup \cdots \cup \mathcal{L}_l$, where each layer is an independent set and only has edges going out to v_p or its two neighbouring layers.

Let us first focus on the subgraph consisting of the vertices in $\{v_p\} \cup \bigcup_{i=0}^{2\rho-3} \mathcal{L}_i$, where any shortest odd cycle has exactly one vertex per layer (per abuse of notation, we say that $\{v_p\}$ is also a layer in this situation). For convenience of indexing, we rename these layers as $\tilde{\mathcal{L}}_1, \ldots, \tilde{\mathcal{L}}_{2\rho-1}$ where $\tilde{\mathcal{L}}_1 = v_p$ and $\tilde{\mathcal{L}}_i = \mathcal{L}_{i-2}$ for $i > 1$. We now create $2\rho-1$ independent sets on this subgraph in the following way. The first independent set is defined as $U_1 = \tilde{\mathcal{L}}_1 \cup \tilde{\mathcal{L}}_4 \cup \tilde{\mathcal{L}}_6 \cdots \cup \tilde{\mathcal{L}}_{2\rho-2}$, where we take the first layer $\tilde{\mathcal{L}}_1$, skip two before taking the next one and then continue by taking the remaining layers alternatingly (hence always skipping one), see Figure 3.6. Note that the layer following $\tilde{\mathcal{L}}_{2\rho-1}$ is assumed to be $\tilde{\mathcal{L}}_1$. This procedure generates in fact a distinct independent set by starting at $\tilde{\mathcal{L}}_k$ for any $k \in [2\rho - 1]$ and we denote the corresponding independent set by U_k. Notice that each layer is contained in exactly $\rho - 1$ of the constructed independent sets $\{U_k \mid k \in [2\rho - 1]\}$.

We now focus on the subgraph consisting of the vertices in $\bigcup_{i>2\rho-3} \mathcal{L}_i$. We can construct two different independent sets there by taking either the odd or

even indexed layers, i.e.

$$R_1 := \bigcup_{i \text{ odd}, i>2\rho-3} \mathcal{L}_i \quad \text{and} \quad R_2 := \bigcup_{i \text{ even}, i>2\rho-3} \mathcal{L}_i.$$

We now define our final $2\rho - 1$ independent sets on the full graph as:

$$I_k := \begin{cases} U_k \cup R_1 & \text{if } v_p \notin U_k \\ U_k \cup R_2 & \text{if } v_p \in U_k \end{cases} \qquad \forall k \in [2\rho - 1].$$

These are in fact independent sets: in the first case, the first layer in R_1 is $\mathcal{L}_{2\rho-1}$ whereas the last layer in U_k has index at most $2\rho - 3$, meaning that there are no two neighbouring layers. In the second case, since $v_p \in U_k$, we have that $\mathcal{L}_{2\rho-3} \notin U_k$, by construction of U_k. The last layer in U_k thus has index at most $2\rho-4$, whereas the first layer in R_2 is $\mathcal{L}_{2\rho-2}$, meaning again that there are no two neighbouring layers. In addition, there is no edge between v_p and R_2, because the only even indexed layer having edges sent to v_p is $\mathcal{L}_0 = N_A(v_p)$.

We now define our primal solution as

$$y(I_k) = \frac{1}{\rho - 1} \qquad \forall k \in [2\rho - 1],$$

and $y(I) = 0$ for every other independent set $I \in \mathcal{I}$. We now show this is a feasible solution, i.e. that every vertex $v \in V$ belongs to at least $\rho - 1$ independent sets in $\{I_k \mid k \in [2\rho - 1]\}$. For $v \in \{v_p\} \cup \bigcup_{i=0}^{2\rho-3} \mathcal{L}_i$, such a vertex lies by construction in exactly $\rho - 1$ independent sets $\{U_k \mid k \in [2\rho - 1]\}$, and thus also of $\{I_k \mid k \in [2\rho - 1]\}$. For $v \in \bigcup_{i>2\rho-3} \mathcal{L}_i$, if v belongs to an even indexed layer, then it is contained in $\rho - 1$ of the desired independent sets. If it belongs to an odd indexed layer, then it is contained in ρ of them. Therefore,

$$\sum_{I \in \mathcal{I}, v \in I} y_I = \sum_{k=1}^{2\rho-1} y(I_k)\, 1_{\{v \in I_k\}} = \frac{1}{\rho - 1} \sum_{k=1}^{2\rho-1} 1_{\{v \in I_k\}} \geq 1.$$

The objective value of this primal solution is clearly $2 + 1/(\rho - 1)$. We have constructed feasible primal and dual solutions with the same objective value. By weak duality, this finishes the proof of the theorem. □

We now consider the case where $\mathcal{G} = (V, E)$ is a graph with chromatic number $\chi(\mathcal{G}) = 3$.

Theorem 3.7.3. *Let $\mathcal{G} = (V, E)$ be a 3-colorable graph with color classes $V = V_1 \cup V_2 \cup V_3$. Then,*

$$\chi^f(\mathcal{G}) \leq 2 + \min_{i \in \{1,2,3\}} \frac{1}{\rho_i - 1}$$

where $2\rho_i - 1$ is the odd girth of the contracted graph $\mathcal{G}/V_i$ for each $i \in \{1, 2, 3\}$. Moreover, equality holds if one color class only contains one vertex.

Proof:
We prove this theorem by constructing three feasible solutions of value $2+1/(\rho_i-1)$ for each $i \in \{1,2,3\}$ to the primal linear program of the fractional chromatic number on the graph $\mathcal{G}$.

Fix an $i \in \{1,2,3\}$ and consider the graph $\tilde{\mathcal{G}} := \mathcal{G}/V_i = (\tilde{V}, \tilde{E})$ with odd girth $2\rho_i - 1$. We denote the contracted node by $\tilde{v} \in \tilde{V}$. Since this graph is bipartite if we were to remove $\tilde{v}$, we know that its fractional chromatic number is equal to $2 + 1/(\rho_i - 1)$ by Theorem 3.7.2. Let $\{\tilde{I}_k, k \in [2\rho_i - 1]\}$ be the independent sets in the support of the optimal primal solution of the graph $\tilde{\mathcal{G}}$ constructed in the proof of this theorem. For each of these independent sets, we extend them to the original graph in the following way:

$$I_k = \begin{cases} \tilde{I}_k & \text{if } \tilde{v} \notin \tilde{I}_k \\ (\tilde{I}_k \setminus \tilde{v}) \cup V_i & \text{if } \tilde{v} \in \tilde{I}_k. \end{cases}$$

In words, if $\tilde{v}$ happens to belong to $\tilde{I}_k$, we replace it by V_i to get a valid independent set in the original graph. Assigning fractional value $y(I_k) = 1/(\rho_i - 1)$ for every $k \in [2\rho_i - 1]$ yields a feasible primal solution with objective value $2 + 1/(\rho_i - 1)$. Since we can do this for every $i \in \{1,2,3\}$, and the optimal minimum value of the primal linear program is at most the objective value of any of these feasible solutions, the proof is finished.

Moreover, this upper bound is in fact tight, since it holds with equality when one of the color classes only contains one vertex by Theorem 3.7.2. □

It is now straightforward to extend this result for the integrality gap by Theorem 3.7.1.

Corollary 3.7.4. *Let $\mathcal{G} = (V, E)$ be a 3-colorable graph with color classes $V = V_1 \cup V_2 \cup V_3$. The integrality gap $IG(\mathcal{G})$ of the standard linear programming relaxation $P(\mathcal{G})$ satisfies:*

$$IG(\mathcal{G}) \leq 1 + \min_{i \in \{1,2,3\}} \frac{1}{2\rho_i - 1}$$

where $2\rho_i - 1$ is the odd girth of the contracted graph $\mathcal{G}/V_i$ for each $i \in \{1,2,3\}$. Moreover, equality holds if one color class only contains one vertex.

Chapter 4

Online matching on 3-uniform hypergraphs

In this chapter, we study an online matching problem on 3-uniform hypergraphs under adversarial vertex arrivals. It is straightforward to see that the greedy algorithm obtains a competitive ratio of 1/3 for this problem by adapting Algorithm 2.5.1 and its analysis in Theorem 2.5.2.

Our main result here is to settle the fractional version of this problem. We first provide an algorithm achieving a competitive ratio of $(e-1)/(e+1) \approx 0.462$. As our main contribution, we then show that this algorithm is in fact optimal by constructing an adversarial instance proving a matching upper bound. We moreover provide a randomized integral algorithm beating the greedy algorithm when the online nodes have bounded degree.

4.1 Introduction

Online matching is a classic problem in the field of online algorithms. It was first introduced in the seminal work of Karp, Vazirani and Vazirani [KVV90], who considered the bipartite version with one-sided vertex arrivals. In this setting, we are given a bipartite graph where vertices on one side are known in advance (offline), and vertices on the other side arrive sequentially (online). When an online vertex arrives, it reveals its incident edges, at which point the algorithm must decide how to match it (or not) irrevocably. The goal is to maximize the cardinality of the resulting matching. Karp et al. [KVV90] gave an elegant randomized algorithm Ranking, which achieves the optimal competitive ratio of $1 - 1/e$.

In certain applications, each offline vertex may be matched more than once. Examples include matching online jobs to servers, or matching online impressions to advertisers. This is the online b-matching model of Kalyanasundaram and Pruhs [KP00b], where $b \geq 1$ is the maximum number of times an offline vertex can be matched. As b and the number of online vertices tend to infinity, it in

turn captures the fractional relaxation of the Karp et al. [KVV90] model. This means that the algorithm is allowed to match an online node fractionally to multiple neighbours, as long as the total load on every vertex does not exceed 1. For this problem, it is known that the deterministic algorithm BALANCE (or WATER-FILLING) achieves the optimal competitive ratio of $1 - 1/e$.

4.1.1 Online hypergraph matching

The online bipartite matching problem can be naturally generalized to hypergraphs as follows. For $k \geq 2$, let $\mathcal{H} = (V, W, H)$ be a k-uniform hypergraph with offline vertices V, online vertices W and hyperedges H. Every hyperedge $h \in H$ contains $k-1$ elements from V and 1 element from W. Just like before, the online vertices arrive sequentially with their incident hyperedges, and the goal is to select a large matching, i.e., a set of disjoint hyperedges. The greedy algorithm is $1/k$-competitive. On the other hand, no integral algorithm can be $2/k$-competitive [1].

For the *fractional* version of the problem, Buchbinder and Naor [BN09] gave a deterministic algorithm which is $\Omega(1/\log k)$-competitive. They also constructed an instance showing that any algorithm is $O(1/\log k)$-competitive. In fact, their results apply to the more general setting of online packing linear program (LP), in which variables arrive sequentially. In the context of hypergraphs, this means that the hyperedges arrive sequentially. Note that for k-uniform hypergraphs, there is a trivial reduction from this *edge-arrival* model to our *vertex-arrival* model on $(k+1)$-uniform hypergraphs, by adding degree 1 online nodes.

The aforementioned results show that asymptotically, both integral and fractional versions of the online matching problem on k-uniform hypergraphs are essentially settled (up to constant factors). However, our understanding of the problem for small values of k (other than $k = 2$) remains poor. Many applications of online hypergraph matching in practice have small values of k. For instance, in ride-sharing and on-demand delivery services [PSST22], $k-1$ represents the capacity of service vehicles, which is often small. Another example is network revenue management problems [MRST20]. In this setting, given a collection of limited resources, a sequence of product requests arrive over time. When a product request arrives, we have to decide whether to accept it irrevocably. Accepting a product request generates profit, but also consumes a certain amount of each resource. The goal is to devise a policy which maximizes profit. In this context, $k-1$ represents the maximum number of resources used by a product. As Ma et al. [MRST20] noted, many of these problems have small values of k. In airlines, for example, $k-1$ corresponds to the maximum number of flight legs included in an itinerary, which usually does not exceed two or three.

[1] In [TU24], it is shown that no algorithm can be $(2 + f(k))/k$-competitive for some positive function f with $f(k) = o(1)$. In Section 4.7, we give a simple construction showing that no integral algorithm can be $2/k$-competitive.

4.1.2 Our contributions

Motivated by the importance of online hypergraph matching for small values of k, we focus on 3-uniform hypergraphs, with the goal of obtaining tighter bounds. Our main result is a tight competitive ratio for the fractional version of this problem.

Theorem 4.1.1. *For the online fractional matching problem on* 3*-uniform hypergraphs, there is a deterministic* $(e-1)/(e+1)$*-competitive algorithm. Furthermore, every algorithm is at most* $(e-1)/(e+1)$*-competitive.*

The deterministic algorithm in Theorem 4.1.1 belongs to the class of WATER-FILLING algorithms. It uses the function $f(x) \coloneqq e^x/(e+1)$ to decide which hyperedges receive load. In particular, for every online vertex w, the incident hyperedges $h = \{u, v, w\} \in \delta(w)$ which minimize $\phi(h) \coloneqq f(x(\delta(u))) + f(x(\delta(v)))$ receive load until $\phi(h) \geq 1$.

Our main contribution is proving a matching upper bound in Theorem 4.1.1. For this, it suffices to consider deterministic algorithms because every randomized algorithm induces a deterministic fractional algorithm with the same expected value. This, in turn, allows us to construct an instance which is adaptive to the actions of the algorithm. The key idea is to combine two hard instances for online matching on *bipartite graphs* [KVV90, GKM$^+$19].

We start with the instance in [GKM$^+$19], designed for the edge-arrival model. In this instance, edge arrivals are grouped into phases, such that the size of an online maximum matching increases by one per phase. At the end of every phase, as long as the total fractional value on the revealed edges exceeds a certain threshold, the next phase begins. Otherwise, the instance terminates. For our purpose, we want a more fine-grained control over the actions of the algorithm. So, we apply thresholding at the node level instead, based on fractional degrees, to determine which nodes become incident to the edges arriving in the next phase.

In our construction, we will have multiple copies of this modified edge-arrival instance. The edges in these instances are connected to the online nodes to form hyperedges. The way in which they are connected is inspired by the instance in [KVV90], originally designed for the vertex-arrival model. The idea behind this vertex-arrival instance is to obfuscate the partners of the online nodes in an offline maximum matching, which is also applicable in our setting.

Our next result concerns the online integral matching problem on k-uniform hypergraphs. We show that one can do better than the greedy algorithm if the online nodes have bounded degree. It is achieved by the simple algorithm RANDOM: for every online vertex w, uniformly select a hyperedge among all the hyperedges incident to w which are disjoint from the current matching.

Theorem 4.1.2. *For the online matching problem on* k*-uniform hypergraphs where online vertices have maximum degree* d*, the competitive ratio of* RANDOM

is at least

$$\min\left(\frac{1}{k-1}, \frac{d}{(d-1)k+1}\right).$$

Note that in Theorem 4.1.2, the first term is at most the second term if and only if $d \leq k-1$. Moreover, RANDOM is at least as good as the greedy algorithm, since the latter is $1/k$-competitive. For 3-uniform hypergraphs, the bound becomes $1/2$ for $d \leq 2$ and $1/(3-2/d)$ otherwise, thus interpolating between $1/3$ and $1/2$. Note that for $d \leq 2$, the bound is optimal, since the online matching problem on graphs under edge arrivals is a special case of this setting (with $k = 3, d = 1$), for which an upper bound of $1/2$ is known even against fractional algorithms on bipartite graphs [GKM$^+$19].

Since every randomized algorithm for integral matching induces a deterministic algorithm for fractional matching, the upper bound of $(e-1)/(e+1) \approx 0.4621$ in Theorem 4.1.1 also applies to the integral problem on 3-uniform hypergraphs. However, the best known lower bound is $1/3$, given by the greedy algorithm. An interesting question for future research is whether there exists an integral algorithm better than greedy on 3-uniform hypergraphs.

4.1.3 Related work

Since the online matching problem was introduced in [KVV90], it has garnered a lot of interest, leading to extensive follow-up work. We refer the reader to the excellent survey by Mehta [Meh13] for navigating this rich literature. The original analysis of RANKING [KVV90] was simplified in a series of papers [BM08, DJK13, GM08, EFFS21]. Many variants of the problem have been studied, such as the online b-matching problem [KP00b], and its extension to the AdWords problem [BJN07, DJ12, HZZ20, MSVV07]. Weighted generalizations have been considered, e.g., vertex weights [AGKM11, HTWZ19] and edge weights [FHTZ22]. Weakening the adversary by requiring that online nodes arrive in a random order has also been of interest [KMT11, MY11, KRTV13]. Another line of research explored more general arrival models such as two-sided vertex arrival [WW15], general vertex arrival [GKM$^+$19], edge arrival [BST19, GKM$^+$19], and general vertex arrival with departure times [HKT$^+$20, HTWZ20, ABD$^+$23].

In contrast, the literature on the online hypergraph matching problem is relatively sparse. Most work has focused on stochastic models, such as the random-order model. Korula and Pal [KP09] first studied the edge-weighted version under this model. For k-uniform hypergraphs, they gave an $\Omega(1/k^2)$-competitive algorithm. This was subsequently improved to $\Omega(1/k)$ by Kesselheim et al. [KRTV13]. Ma et al. [MRST20] gave a $1/k$-competitive algorithm under 'nonstationary' arrivals. Pavone et al. [PSST22] studied online hypergraph matching with delays under the adversarial model. At each time step, a vertex arrives, and it will depart after d time steps. A hyperedge is revealed once all of its vertices have

arrived. Note that their model is incomparable to ours because every vertex has the same delay d.

In the prophet IID setting, every online node has a weight function which assigns weights to its incident hyperedges, and these functions are independently sampled from the same distribution. For this problem, [MSV24] gave a $O(\log(k)/k)$ upper bound on the competitive ratio. We refer to [MSV24] for an overview of known results in related settings.

Hypergraph matching on k-uniform hypergraphs is a well-studied problem in the offline setting. It is NP-hard to approximate within a factor of $\Omega(\log(k)/k)$ [HSS06]. Moreover, the factor between the optimal solution and the optimal value of the natural LP relaxation is at least $1/(k-1+1/k)$ [CL12].

A special case that has also been studied is the restriction to k-partite graphs, where the vertices are partitioned into k sets and every hyperedge contains exactly one vertex from each set. This setting is called *k-dimensional matching*, and the optimal solution is known to be at least $1/(k-1)$ times the optimal value of the standard LP relaxation [CL12]. For $k=3$, the best known polynomial time approximation algorithm gives a $(3/4-\varepsilon)$-approximation [Cyg13].

4.1.4 Chapter organization

In Section 4.2, we give the necessary preliminaries and discuss notation. Section 4.3 presents the optimal primal-dual fractional algorithm for 3-uniform hypergraphs, which shows the first part of Theorem 4.1.1. Section 4.4 complements this with a tight upper bound, proving the second part of Theorem 4.1.1. The proof of Theorem 4.1.2 is shown in Section 4.5.

4.2 Preliminaries

Given a hypergraph $\mathcal{H}=(V,H)$ with vertex set V and hyperedge set H, the maximum matching problem involves finding a maximum cardinality subset of disjoint hyperedges. The canonical primal and dual LP relaxations for this problem are respectively given by:

$$\begin{aligned} \max \sum_{h\in H} x_h & \\ \sum_{h\in\delta(v)} x_h \le 1 \quad & \forall v\in V \\ x_h \ge 0 \quad & \forall h\in H \end{aligned} \qquad\qquad \begin{aligned} \min \sum_{v\in V} y_v & \\ \sum_{v\in h} y_v \ge 1 \quad & \forall h\in H \\ y_v \ge 0 \quad & \forall v\in V. \end{aligned}$$

We denote by $\mathsf{OPT}_{\mathsf{LP}}(\mathcal{H})$ the offline optimal value of these two LPs. We denote by $\mathsf{OPT}(\mathcal{H})$ the objective value of an offline optimal integral solution to the primal LP, which clearly satisfies $\mathsf{OPT}(\mathcal{H}) \le \mathsf{OPT}_{\mathsf{LP}}(\mathcal{H})$.

The online matching problem on k-uniform hypergraphs under vertex arrivals is defined as follows. An instance consists of a k-uniform hypergraph $\mathcal{H} = (V, W, H)$, where V is the set of offline nodes and $W = (w_1, w_2, \ldots)$ is the sequence of online nodes. The ordering of W corresponds to the arrival order of the online nodes. Every hyperedge $h \in H$ has exactly one node in W and $k-1$ nodes in V. When an online node $w \in W$ arrives, its incident hyperedges $\delta(w)$ are revealed. A fractional algorithm is allowed to irrevocably increase x_h for every $h \in \delta(w)$, whereas an integral algorithm is allowed to irrevocably pick one of these hyperedges, i.e., setting $x_h = 1$ for some $h \in \delta(w)$.

Given an algorithm $\mathcal{A}$ and an instance $\mathcal{H}$, we denote by $\mathcal{V}(\mathcal{A}, \mathcal{H}) := \sum_{h \in H} x_h$ the value of the (fractional) matching obtained by $\mathcal{A}$ on $\mathcal{H}$. An integral algorithm is *ρ-competitive* if for any instance $\mathcal{H}$, $\mathcal{V}(\mathcal{A}, \mathcal{H}) \geq \rho\, \mathsf{OPT}(\mathcal{H})$. Similarly, a fractional algorithm is *ρ-competitive* if for any instance $\mathcal{H}$, $\mathcal{V}(\mathcal{A}, \mathcal{H}) \geq \rho\, \mathsf{OPT}_{\mathsf{LP}}(\mathcal{H})$.

In this thesis, we focus on 3-uniform hypergraphs. For a 3-uniform instance $\mathcal{H} = (V, W, H)$, we denote by $\Gamma(\mathcal{H}) = (V, E)$ the graph on the offline nodes with edge set

$$E := \Big\{(u, v) \in V \times V, \quad \exists w \in W \text{ s.t. } \{u, v, w\} \in H\Big\}. \tag{4.2.1}$$

We remark that $\Gamma(\mathcal{H})$ is not a multigraph. In particular, an edge $(u, v) \in E$ can have several hyperedges in H containing it. A fractional matching x on the hyperedges H naturally induces a fractional matching x' on the edges E, i.e., $x'_e = \sum_{h : e \subseteq h} x_h$ for every $e \in E$. The value obtained by an algorithm $\mathcal{A}$ can thus also be counted as $\mathcal{V}(\mathcal{A}, \mathcal{H}) = \sum_{h \in H} x_h = \sum_{e \in E} x'_e$. For an offline node $u \in V$, we denote its *load* (or *fractional degree*) as $\ell_u = x(\delta(u)) \in [0, 1]$.

4.3 Optimal fractional algorithm for 3-uniform hypergraphs

In this section, we present a primal-dual algorithm for the online fractional matching problem on 3-uniform hypergraphs under vertex arrivals. This algorithm will turn out to be optimal with a tight competitive ratio of $(e-1)/(e+1) \approx 0.4621$. We define the following distribution function $f : [0, 1] \to [0, 1]$:

$$f(x) := \frac{e^x}{e+1}. \tag{4.3.1}$$

When an online node w arrives, our algorithm chooses to uniformly increase the primal variables of the hyperedges $\{u, v, w\}$ for which $f(x(\delta(u))) + f(x(\delta(v)))$ is minimal. We note that this belongs to the class of *water-filling* algorithms [KP00b]. For this reason, we define the *priority* of a hyperedge $h = \{u, v, w\}$ as:

$$\phi(h) := f(x(\delta(u))) + f(x(\delta(v))). \tag{4.3.2}$$

Figure 4.1 shows the possible values of $x(\delta(u))$ and $x(\delta(v))$ such that $\phi(h) \leq 1$. We now present the algorithm.

Algorithm 4.3.1 Water-filling fractional algorithm for 3-uniform hypergraphs

Input : 3-uniform hypergraph $\mathcal{H} = (V, W, H)$ with online nodes W.
Output : Fractional matching $x \in [0,1]^H$

when $w \in W$ **arrives with** $\delta(w) \subseteq H$**:**
set $x_h = 0$ for every $h \in \delta(w)$
increase x_h for every $h = \{u, v, w\} \in \arg\min_{h \in \delta(w)}\{\phi(h)\}$ at rate 1
increase y_u and y_v at rates $f(x(\delta(u)))$ and $f(x(\delta(v)))$
increase y_w at rate $1 - f(x(\delta(u))) - f(x(\delta(v)))$
until $x(\delta(w)) = 1$ **or** $\phi(h) \geq 1$ for every $h \in \delta(w)$.
return x

Theorem 4.3.1. *Algorithm 4.3.1 is* $(e-1)/(e+1)$*-competitive for the online fractional matching problem on* 3*-uniform hypergraphs.*

Proof:
We first show that the algorithm produces a feasible primal solution. Note that the fractional value of a hyperedge h is only being increased if $\phi(h) \leq 1$. If $x(\delta(v)) = 1$ for some offline node v then for any hyperedge $h \ni v$, we have:

$$\phi(h) = f(x(\delta(u))) + f(x(\delta(v))) \geq f(1) + f(0) = \frac{e+1}{e+1} = 1,$$

where u denotes the second offline node belonging to h. The value of the hyperedge h will thus not be increased anymore, proving the feasibility of the primal solution.

In order to prove the desired competitive ratio, we show that the primal-dual solutions constructed during the execution of the algorithm satisfy:

$$\mathcal{V}(\mathcal{A}) = \sum_{h \in H} x_h = \sum_{v \in V \cup W} y_v \qquad \text{and} \tag{4.3.3}$$

$$\sum_{v \in h} y_v \geq \rho \qquad \forall h \in H. \tag{4.3.4}$$

This is enough to imply the desired competitiveness of our algorithm, since $y/\rho \in \mathbb{R}_+^V$ is then a feasible dual solution, giving:

$$\mathcal{V}(\mathcal{A}) \geq \sum_{v \in V \cup W} y_v \geq \rho \, \mathsf{OPT}_{\mathsf{LP}}.$$

Note that (4.3.3) holds at the start of the algorithm. Let us fix a hyperedge $h = \{u, v, w\} \in H$. When x_h is continuously being increased at rate one, the

duals on the incident nodes y_u, y_v and y_w are being increased at rate $f(x(\delta(u)))$, $f(x(\delta(v)))$ and $1-f(x(\delta(u)))-f(x(\delta(v)))$ respectively. Observe that these rates sum up to one. Hence, $\mathcal{V}(\mathcal{A}) = \sum_{h\in H} x_h$ and $\sum_{v\in V\cup W} y_v$ are increased at the same rate, meaning that (4.3.3) holds at all times during the execution of the algorithm.

We now show that (4.3.4) holds at the end of the execution of the algorithm. Let us fix an online node $w \in W$. For a given hyperedge $h \in \delta(w)$, note that the algorithm only stops increasing x_h, as soon as either $\phi(h) \geq 1$ or $x(\delta(w)) = 1$ is reached. We distinguish these two cases for the analysis.

Let us first focus on the first case, meaning that $\phi(h) \geq 1$ has been reached for every $h \in \delta(w)$. Consider an arbitrary $h = \{u, v, w\} \in \delta(w)$. For every unit of increase in $x(\delta(u))$, y_u will have been increased by $f(x(\delta(u)))$. If we denote by $\ell_u := x(\delta(u))$ and $\ell_v := x(\delta(v))$ the fractional loads on u and v after the last increase on the hyperedges adjacent to w, then:

$$y_u = \int_0^{\ell_u} f(s)ds = f(\ell_u) - f(0) \quad \text{and} \quad y_v = \int_0^{\ell_v} f(s)ds = f(\ell_v) - f(0), \tag{4.3.5}$$

where we have used the fact that f is an antiderivative of itself. Therefore,

$$\begin{aligned} y_u + y_v + y_w \geq y_u + y_v &= f(\ell_u) - f(0) + f(\ell_v) - f(0) \\ &= \phi(h) - 2f(0) \geq 1 - 2f(0) = \frac{e-1}{e+1}. \end{aligned}$$

Suppose now that $x(\delta(w)) = 1$ has been reached. In particular, this means that for each $\{u, v, w\} \in \delta(w)$, the rate at which y_w was increased must have been at least $1 - f(\ell_u) - f(\ell_v)$ at all times, where ℓ_u and ℓ_v denote the fractional loads on u and v after that the algorithm has finished increasing the edges incident to the online node w. Hence, we have:

$$y_w \geq 1 \cdot (1 - f(\ell_u) - f(\ell_v)).$$

By using (4.3.5) we see that:

$$y_u + y_v + y_w \geq f(\ell_u) + f(\ell_v) - 2f(0) + (1 - f(\ell_u) - f(\ell_v)) = 1 - 2f(0) = \frac{e-1}{e+1}.$$

This proves (4.3.4), and thus completes the proof of the theorem. □

4.4 Tight upper bound for 3-uniform hypergraphs

We now prove the second part of Theorem 4.1.1, i.e., every algorithm is at most $(e-1)/(e+1)$-competitive for the online fractional matching problem on 3-uniform hypergraphs under vertex arrivals.

4.4.1 Overview of the construction

We construct an adversarial instance that is adaptive to the behaviour of the algorithm. The main idea is to combine the vertex-arrival instance of Karp et al. [KVV90] and the edge-arrival instance of Gamlath et al. [GKM+19] for bipartite graphs.

We start by giving a high-level overview of the construction. The offline vertices of the hypergraph are partitioned into m sets $C_1, \dots, C_m$, which we call *components*. Each component will induce a bipartite graph with bipartition $C_i = U_i \cup V_i$, where $|U_i| = |V_i| = T$.

The instance consists of T phases. In each phase $t \in \{1, \dots, T\}$, the adversary first selects a bipartite matching $\mathcal{M}_i^{(t)}$ on each component C_i. Taking the union of these matchings gives a larger matching on the offline nodes: $\mathcal{M}^{(t)} := \bigcup_{i=1}^{m} \mathcal{M}_i^{(t)}$.

After selecting the matching $\mathcal{M}^{(t)}$ at phase t, the adversary selects the online nodes, with their incident hyperedges, arriving in that phase. The set of online nodes arriving in phase t is denoted by $W^{(t)}$. Each node $w \in W^{(t)}$ connects to a subset of edges $E(w) \subseteq \mathcal{M}^{(t)}$, meaning that the hyperedges incident to w are $\{\{w\} \cup e : e \in E(w)\}$.

We briefly explain how the matchings $\mathcal{M}_i^{(t)}$ are constructed and how the edges $E(w)$ are picked:

1. On each component C_i, the matching $\mathcal{M}_i^{(t)}$ is constructed based on the behaviour of the algorithm in phase $t-1$. It draws inspiration from the edge-arrival instance in [GKM+19], together with the function $f(x) = e^x/(e+1)$ defined in (4.3.1). The exact construction is described in Section 4.4.3 and illustrated in Figures 4.2 and 4.3.

2. For every online node $w \in W^{(t)}$, the edge set $E(w) \subseteq \mathcal{M}^{(t)}$ is selected based on the behaviour of the algorithm during phase t. This part can be seen as incorporating the vertex-arrival instance in [KVV90]. The exact construction is described in Section 4.4.5 and illustrated in Figure 4.6.

To summarize, the instance is a hypergraph $\mathcal{H} = (V, W, H)$ with offline nodes V, online nodes W and hyperedges H given by

$$V := \bigcup_{i=1}^{m} C_i = \bigcup_{i=1}^{m} U_i \cup V_i \qquad W := \bigcup_{t=1}^{T} W^{(t)} \qquad H := \bigcup_{t=1}^{T} \bigcup_{w \in W^{(t)}} \{\{w\} \cup e : e \in E(w)\}.$$

4.4.2 Assumptions on the algorithm

To simplify the construction and analysis of our instance, we will make two assumptions on the algorithm. First, we need the following definition, which relates the behaviour of an algorithm to the priority function ϕ defined in (4.3.2).

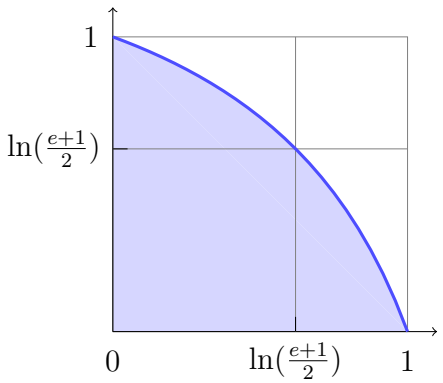


Figure 4.1: An illustration of the region $R = \{(a, b) \in [0, 1]^2 : f(a) + f(b) \leq 1\}$. The symmetric point at the boundary of the region has both coordinates $\ln((e + 1)/2) \approx 0.62$. When an online node w arrives, a threshold respecting algorithm ensures that the fractional matching x satisfies $(x(\delta(u)), x(\delta(v))) \in R$ for every hyperedge $h = \{u, v, w\} \in \delta(w)$ with $x_h > 0$ at the end of that iteration.

Definition 4.4.1. Fix $\varepsilon \geq 0$. Let x be the fractional solution given by an algorithm $\mathcal{A}$ after the arrival of an online node w. We say that $\mathcal{A}$ is *ε-threshold respecting* on w if $\phi(h) = \sum_{v \in h \setminus \{w\}} f(x(\delta(v))) \leq 1 + \varepsilon$ for all incident hyperedges $h \in \delta(w)$ with $x_h > 0$. We also call $\mathcal{A}$ *threshold respecting* if $\varepsilon = 0$.

Remark 4.4.2. We emphasize that the property in Theorem 4.4.1 only needs to hold for the fractional solution x after w has arrived, and before the arrival of the next online node. In particular, it is possible that $\phi(h) > 1 + \varepsilon$ in later iterations. For a hyperedge $h = \{u, v, w\} \in \delta(w)$, Figure 4.1 shows the possible values of $x(\delta(u))$ and $x(\delta(v))$ such that $\phi(h) \leq 1$.

The two assumptions that we make are the following. In Section 4.6, we show that they can be made without loss of generality.

1. The algorithm is *ε-threshold respecting* on all online nodes in the first $T - 1$ phases for some arbitrarily small $\varepsilon > 0$.

2. The algorithm is *symmetric* on each component $C_i = U_i \cup V_i$. In particular, for every $t \in \{1, \ldots, T\}$, the tth vertices of U_i and V_i have the same fractional degrees throughout the execution of the algorithm.

4.4.3 Constructing the matching $\mathcal{M}^{(t)}$

In this section, we construct the matching $\mathcal{M}_i^{(t)}$ for every component $i \in [m]$ and phase $t \in [T]$. We will only describe the matchings for a single component C_i, i.e., $\mathcal{M}_i^{(1)}, \mathcal{M}_i^{(2)}, \ldots, \mathcal{M}_i^{(T)}$, because the same construction applies to other

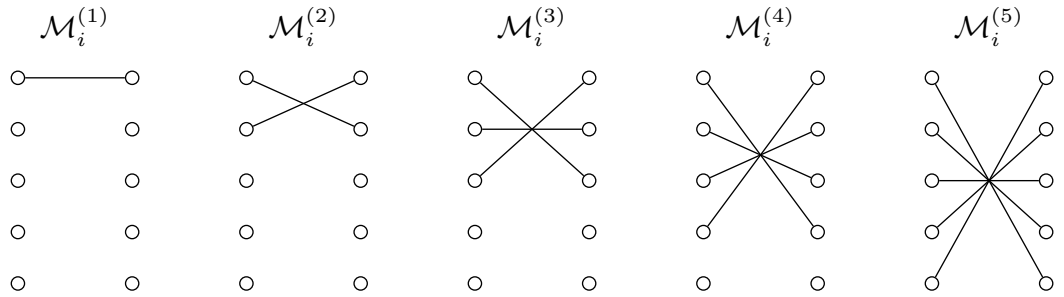


Figure 4.2: The partial matchings $\mathcal{M}_i^{(t)}$ if the fractional algorithm ensures that every edge reaches the threshold at the end of every phase, meaning that $f(\ell_u^{(t)}) + f(\ell_v^{(t)}) \geq 1$ for every $(u, v) \in \mathcal{M}_i^{(t)}$. The maximum matching at the end of phase 5 has size five and consists of $\mathcal{M}_i^{(5)}$.

components. Intuitively, the value obtained by the algorithm in the first $T - 1$ phases is already limited by Assumption 1. So, the goal of this construction is to prevent the algorithm from gaining too much value in the last phase T. In particular, we will show that it can only obtain $O(\sqrt{T}) + \varepsilon\ O(T^2)$ in the last phase on every component C_i.

The matchings $\mathcal{M}_i^{(1)}, \ldots, \mathcal{M}_i^{(T)}$ are adaptive to the behaviour of the algorithm in every phase. It is essentially the instance of [GKM+19] with our threshold function incorporated. The vertex set of these matchings is on a bipartite graph, with T nodes on both sides of the bipartition. Let us denote this bipartition as $U_i = \{1, \ldots, T\}$ and $V_i = \{1, \ldots, T\}$. We index them the same way due to the symmetry assumption of the algorithm (Assumption 2). Each matching $\mathcal{M}_i^{(t)}$ satisfies the invariant that $(u, v) \in \mathcal{M}_i^{(t)}$ if and only if $(v, u) \in \mathcal{M}_i^{(t)}$.

For an offline node $u \in V$, we denote its *load* (or *fractional degree*) at the end of phase t as $\ell_u^{(t)} = x^{(t)}(\delta(u)) \in [0, 1]$, where $x^{(t)}$ is the fractional matching generated by the algorithm at the end of phase t.

- $\mathcal{M}_i^{(1)}$ is a matching of size one that consists of the single edge $(1, 1)$.
- At the end of phase t, we will call a node *active* if it is incident to an edge $e = (u, v) \in \mathcal{M}_i^{(t)}$ satisfying $\phi(e) = f(\ell_u^{(t)}) + f(\ell_v^{(t)}) \geq 1$. All other nodes are said to be *inactive* and will not be used in any of the matchings of later phases. Let $\sigma_t(1) < \sigma_t(2) < \ldots < \sigma_t(r_t)$ be the active nodes in U_i at the end of phase t, where r_t denotes the number of such active nodes. By the aforementioned invariant and Assumption 2, the active nodes in V_i are also $\sigma_t(1) < \sigma_t(2) < \ldots < \sigma_t(r_t)$.
- The matching at phase $t + 1$ is then of size $r_t + 1$ and is defined as:

$$\mathcal{M}_i^{(t+1)} := \Big\{ \Big(\sigma_t(k), \sigma_t(r_t + 2 - k) \Big), k \in \{1, \ldots, r_t + 1\} \Big\},$$

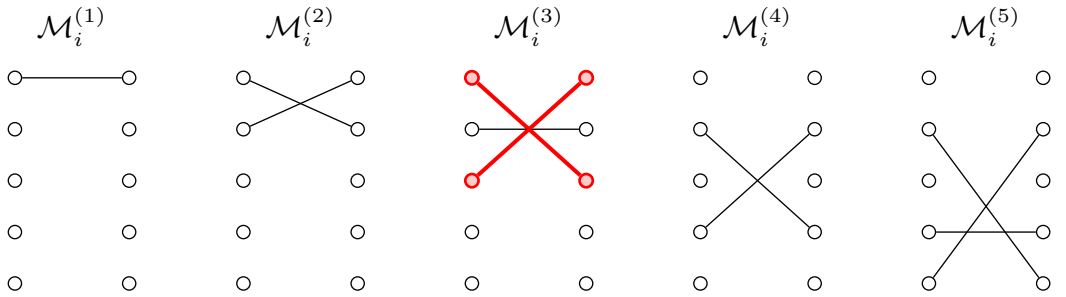


Figure 4.3: In this example, the algorithm does not increase the edge $(1,3)$, and thus by symmetry the edge $(3,1)$, up to the threshold during phase $t=3$. Hence, $f(\ell_1^{(3)}) + f(\ell_3^{(3)}) < 1$ and nodes 1 and 3 become inactive from that point on. The maximum matching at the end of phase 5 still has size five and consists of $\mathcal{M}_i^{(5)}$, in addition to the two edges $(1,3)$ and $(3,1)$ that are below the threshold.

where we define $\sigma_t(r_t+1) := t+1$ for convenience. In particular, note that $t+1 \in U_i$ and $t+1 \in V_i$ are two fresh nodes with zero load, which are always part of the matching $\mathcal{M}_i^{(t+1)}$, but not part of any matching from a previous phase. Clearly, the invariant is maintained. Figures 4.2 and 4.3 illustrate the construction.

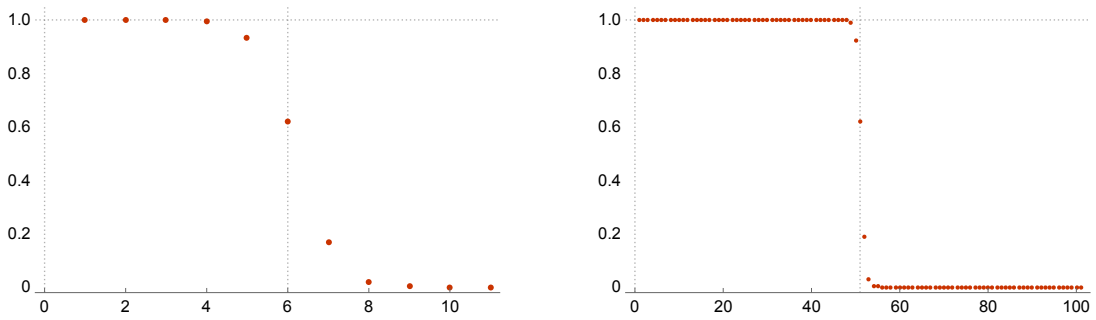
Let us denote $q_t := (r_t+1)/2$. Observe that the nodes $\sigma_t(k) \in U_i$ and $\sigma_t(k) \in V_i$ for every $k \in \{1, \dots, \lceil q_t \rceil\}$ form a vertex cover of the matching $\mathcal{M}_i^{(t+1)}$, meaning that every edge of the matching in phase $t+1$ is covered by one of these active nodes at phase t. Intuitively, this construction ensures that as t gets large, these nodes have a high fractional degree. Consequently, the algorithm does not have a lot of room to increase the fractional value on any edge of $\mathcal{M}_i^{(t+1)}$, due to the degree constraints. In order to upper bound the value that the algorithm can get in phase $t+1$, we will thus lower bound the fractional degree of the active nodes $\sigma_t(i)$ for $i \in \{1, \dots, \lceil q_t \rceil\}$. For this reason, we define:

$$\ell(t,i) := x^{(t)}\Big(\delta(\sigma_t(i))\Big) = \sum_{e \in \delta(\sigma_t(i))} x_e^{(t)}.$$

In words, this is the fractional degree of the i^{th} active node at the end of phase t. One can now see $\{\ell(t,i)\}_{t,i}$ as a process with two parameters, which depends on the behaviour of the algorithm. To analyze this process, we will relate it to the CDF of the binomial distribution $B(t, 1/2)$. We will in fact show that

$$\sum_{i=1}^{\lceil q_{T-1} \rceil} 2(1 - \ell(T-1,i)) = O(\sqrt{T}) + \varepsilon\, O(T^2). \tag{4.4.1}$$

Since the left-hand side is the residual capacity of the vertex cover of $\mathcal{M}_i^{(T)}$, this will yield an upper bound on the value obtained by the algorithm in component



(a) $t = 11, q_t = 6$ (b) $t = 101, q_t = 51$

Figure 4.4: Plot of the $\ell(t, i)$ process for two different values of t if the algorithm exactly matches the threshold at every phase. Observe that, since t is odd, $(\sigma_t(q_t), \sigma_t(q_t)) \in \mathcal{M}_i^{(t)}$ with the load of node $\sigma_t(q_t)$ staying at $\ln((e+1)/2) \approx 0.62$.

C_i during the last phase. For intuition, Figure 4.4 provides an example of $\ell(t, i)$ if the algorithm exactly reaches the threshold for every edge.

4.4.4 Bound for the last phase T

In this section, we prove the following theorem by showing (4.4.1).

Theorem 4.4.3. *During the last phase T, the value gained by the algorithm in each component C_i is at most $O(\sqrt{T}) + \varepsilon\, O(T^2)$.*

In order to be able to get a lower bound on $\ell(t, i)$, we now relate it to a process which is simpler to analyze, defined as follows on $\mathbb{N} \times \mathbb{Z}/2$:

$$\psi(t, y) = \Pr_{X \sim B(t, \frac{1}{2})}\left[X < \frac{t}{2} + y\right] + \frac{1}{2} \Pr_{X \sim B(t, \frac{1}{2})}\left[X = \frac{t}{2} + y\right],$$

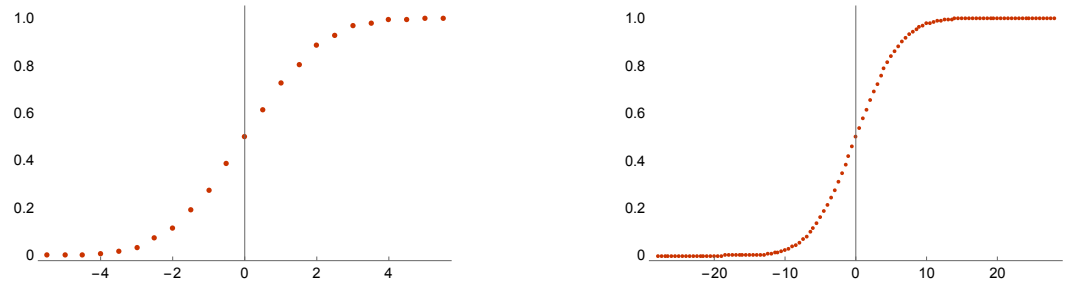
where $B(t, \frac{1}{2})$ is the binomial distribution with parameters t and $\frac{1}{2}$ (see Figure 4.5 for an illustration). An important property of this function that will help us prove Theorem 4.4.3 is the following upper bound:

$$\sum_{y=0}^{\infty} \left(1 - \psi\left(t, \frac{y}{2}\right)\right) \leq 1 + \frac{1}{2}\sqrt{t} \qquad \forall t \geq 1. \tag{4.4.2}$$

We then relate the process $\ell(t, i)$ to a linear transformation of the process $\psi(t, i)$, by defining:

$$\xi(t, i) := a\, \psi\left(t, q_t - i\right) + b - \varepsilon t,$$

where $a := 2 - 2\ln\Big((e+1)/2\Big) \approx 0.76$ and $b := 2\ln\Big((e+1)/2\Big) - 1 \approx 0.24$. Here, a and b are chosen such that whenever the algorithm exactly hits the threshold



(a) $t = 11$ (b) $t = 101$

Figure 4.5: Plot of $\psi(t, y)$ for two different values of t, the horizontal axis represents $y \in \mathbb{Z}/2$.

for every edge, we have $\xi(t, t/2) = \ln((e+1)/2) = \ell(t, t/2)$ for all even t and $\lim_{t\to\infty} \xi(t, x) = 1 = \lim_{t\to\infty} \ell(t, x)$ for all x. Let us now state the properties we need from the ψ function.

Claim 4.4.1. *The function ψ satisfies the following properties:*

1. *For all t, $\psi(t, \frac{t+1}{2}) = 1$.*
2. *For all t, $\psi(t, y)$ is nondecreasing in y.*
3. *For all t, we have: $\psi(t, 0) = \frac{1}{2}$.*
4. *For all t and y, we have: $\psi(t+1, y) = \frac{1}{2}\psi(t, y - \frac{1}{2}) + \frac{1}{2}\psi(t, y + \frac{1}{2})$.*
5. *For all t, we have $\sum_{y=0}^{\infty}(1 - \psi(t, \frac{1}{2}y)) \leq 1 + \frac{1}{2}\sqrt{t}$.*

Proof:
The first two statements follow directly from the definition. The third statement follows from the symmetry of $B(t, \frac{1}{2})$ around $\frac{t}{2}$. For the fourth statement, let $X \sim B(t, \frac{1}{2})$ and $Y \sim B(1, \frac{1}{2})$ be independent. Then, we have:

$$\begin{aligned}
\psi(t+1, y) &= \Pr\left[X + Y < \frac{t+1}{2} + y\right] + \frac{1}{2}\Pr\left[X + Y = \frac{t+1}{2} + y\right] \\
&= \Pr[Y = 1]\Pr\left[X < \frac{t}{2} + y - \frac{1}{2}\right] + \Pr[Y = 0]\Pr\left[X < \frac{t}{2} + y + \frac{1}{2}\right] \\
&\quad + \frac{1}{2}\Pr[Y = 0]\Pr\left[X = \frac{t}{2} + y + \frac{1}{2}\right] + \frac{1}{2}\Pr[Y = 1]\Pr\left[X = \frac{t}{2} + y - \frac{1}{2}\right] \\
&= \frac{1}{2}\psi\left(t, y - \frac{1}{2}\right) + \frac{1}{2}\psi\left(t, y + \frac{1}{2}\right).
\end{aligned}$$

Now, let us prove the last statement. Let $X \sim B(t, \frac{1}{2})$ and observe that $1 - \psi(t, y) \leq \Pr[X \geq \frac{t}{2} + y]$, leading to:

$$\begin{aligned}\sum_{y=0}^{\infty}(1-\psi(t,\frac{1}{2}y)) &\leq \sum_{y=0}^{\infty}\Pr\left[X-\frac{t}{2}\geq\frac{y}{2}\right] \leq \sum_{y=0}^{\infty}\Pr\left[\left|X-\frac{t}{2}\right|\geq\frac{y}{2}\right]\\ &\leq 1+2\,\mathbb{E}\left[|X-\frac{t}{2}|\right] \leq 1+2\sqrt{\mathbb{E}\left[\left(X-\frac{t}{2}\right)^2\right]}\\ &= 1+2\sqrt{\mathrm{Var}\,[X]} = 1+\frac{1}{2}\sqrt{t}.\end{aligned}$$

where the last inequality follows by Jensen's inequality. □

We need one additional lemma before being able to get a lower bound for the loads of the nodes. As a reminder, r_t is the number of active nodes on each side of the bipartition at the end of phase t and $q_t := (r_t+1)/2$. For every active node $u = \sigma_t(i)$, where $i \leq r_t$, let us define $d_t(u) = |i - q_t|$. This quantity measures the distance, in index, between node u and the active node at position q_t.

Lemma 4.4.4. *Let $u = \sigma_t(i)$ for $i \leq r_t$, then the following holds:*

$$d_t(u) \leq d_{t-1}(u) + \frac{1}{2} \quad \textit{if } i < q_t \qquad \textit{and} \qquad d_t(u) \leq d_{t-1}(u) - \frac{1}{2} \quad \textit{if } i > q_t.$$

Proof:
Let S be the set of indices $i \leq r_{t-1} + 1$ such that $\sigma_{t-1}(i)$ is not active after phase t. Let j be such that $\sigma_{t-1}(j) = u$. Consider the case that $i < q_t$. Let $c = |s \in S : i < s < r_{t-1} + 2 - i|$. We have $d_t(u) = q_t - i = q_{T-1} + \frac{1}{2} - \frac{1}{2}c - i \leq q_{T-1} + \frac{1}{2} - i = d_{t-1}(u) + \frac{1}{2}$. The proof for $i > q_t$ is similar. □

We are now ready to prove the desired lower bound on the loads.

Lemma 4.4.5. *For every $t \in \{1, \ldots, T-1\}$ and $i \in \{1, \ldots, \lceil q_t \rceil\}$, we have*

$$\ell(t, i) \geq \xi(t, i),$$

where $q_t = (r_t + 1)/2$ and r_t is the number of active nodes at the end of phase t, when $r_t \geq 1$.

Proof:
We will prove this statement by induction on t. For the base case, consider $t = 1$. There are two possibilities, either the edge $(1, 1)$ does not make it to the threshold, i.e. $2f(\ell_1^{(1)}) < 1$, in which case $r_t = 0$, $q_t = 0$, and the statement is then trivially satisfied. If the edge $(1, 1)$ makes it to the threshold, then $2f(\ell_1^{(1)}) = 2f(\ell(1, 1)) \geq$

1, which is equivalent to $\ell(1,1) \geq \ln((e+1)/2)$ by definition of $f(x) = e^x/(e+1)$. Observe that in this case $r_t = q_t = 1$, leading to

$$\ell(1,1) \geq \ln((e+1)/2) = \frac{a}{2} + b = a\,\psi(1,0) + b = \xi(1,1) + \varepsilon t \geq \xi(1,1),$$

where we have used the fact that $\psi(1,0) = 1/2$.

Suppose now by induction that the statement holds for $t-1$, let r_t be the active nodes at the end of phase t, let $q_t := (r_t+1)/2$ and consider an arbitrary $i \in \{1, \ldots, \lceil q_t \rceil\}$. Let us first consider the case where $i = q_t = (r_t+1)/2$, which can only occur when r_t is odd. Observe that this means the edge $(\sigma_t(i), \sigma_t(i))$ belongs to the matching $\mathcal{M}_i^{(t)}$ and exceeds the threshold, i.e. $\ell(t,i) \geq \ln((e+1)/2)$. Using the fact that $\psi(t,0) = 1/2$ for all t, $i = q_t$ and the exact same arguments as above, we get

$$\ell(t,i) \geq \ln((e+1)/2) = \frac{a}{2} + b = a\psi(t,0) + b = a\psi(t, q_t - i) + b = \xi(t,i) + \varepsilon t \geq \xi(t,i).$$

Consider now the case where $i < q_t$. Let $e = (u,v) = (\sigma_t(i), \sigma_t(r_t + 1 - i)) \in \mathcal{M}_i^{(t)}$ and observe that e exceeds the threshold, i.e. $f(\ell_u^{(t)}) + f(\ell_v^{(t)}) = f(\ell(t,i)) + f(\ell(t, r_t + 1 - i)) \geq 1$. Let us pick indices j, k such that $u = \sigma_{t-1}(j)$ and $v = \sigma_{t-1}(r_{t-1} + 1 - k)$. Observe that:

$$\ell(t,i) = \ell(t-1,j) + x_e^{(t)} \quad \text{and} \quad \ell(t, r_t + 1 - i) = \ell(t-1, r_{t-1} + 1 - k) + x_e^{(t)}.$$

If $k = 0$, then v has not appeared in any of the prior matchings, so $\ell_v^{(t-1)} = 0$. In particular, we have $f(\ell_v^{(t-1)}) = f(0) = 1 - f(1) \leq 1 + \varepsilon - f(\xi(t-1,k))$.

Otherwise, we have $(\sigma_{t-1}(k), v) \in \mathcal{M}_i^{(t-1)}$ and by using the fact that the algorithm is ε-threshold respecting, we get $f(\ell(t-1,k)) + f(\ell_v^{(t-1)}) \leq 1 + \varepsilon$. By using the inductive hypothesis $\ell(t-1,k) \geq \xi(t-1,k)$, we get

$$f(\ell_v^{(t-1)}) \leq 1 + \varepsilon - f(\xi(t-1,k)).$$

Since edge e exceeds the threshold at the end of phase t, we have $f(\ell_u^{(t-1)} + x_e^{(t)}) + f(\ell_v^{(t-1)} + x_e^{(t)}) \geq 1$, which leads to

$$\begin{aligned} x_e^{(t)} &\geq -\ln\left(f(\ell_u^{(t-1)}) + f(\ell_v^{(t-1)})\right) \geq -\ln\left(1 + \varepsilon - f(\xi(t-1,k)) + f(\ell_u^{(t-1)})\right) \\ &\geq f(\xi(t-1,k)) - f(\ell_u^{(t-1)}) - \varepsilon = f(\xi(t-1,k)) - f(\ell(t-1,j)) - \varepsilon, \end{aligned}$$

where we have used the fact that $f(x) = e^x/(e+1)$ is an increasing function and $\ln(1+x) \geq x$.

Finally, since we have $h'(\ell_u^{(t-1)}) = f(\ell_u^{(t-1)}) \geq h\Big(\ln((e+1)/2)\Big) = 1/2$:

$$\begin{aligned}
\ell(t,i) &= \ell_u^{(t-1)} + x_e^{(t)} \geq \ell_u^{(t-1)} + h'(\ell_u^{(t-1)})\Big(\xi(t-1,k) - \ell(t-1,j)\Big) - \varepsilon \\
&= \frac{1}{2}\ell(t-1,j) + \frac{1}{2}\xi(t-1,k) - \varepsilon \\
&\geq \frac{1}{2}\xi(t-1,j) + \frac{1}{2}\xi(t-1,k) - \varepsilon \\
&= \frac{a}{2}\Big(\psi\Big(t-1, d_{t-1}(u)\Big) + \psi\Big(t-1, d_{t-1}(v)\Big)\Big) + b - \varepsilon t \\
&\geq \frac{a}{2}\Big(\psi\Big(t-1, d_t(u) + \frac{1}{2}\Big) + \psi\Big(t-1, d_t(v) - \frac{1}{2}\Big)\Big) + b - \varepsilon t \\
&= \frac{a}{2}\Big(\psi\Big(t-1, d_t(u) + \frac{1}{2}\Big) + \psi\Big(t-1, d_t(u) - \frac{1}{2}\Big)\Big) + b - \varepsilon t \\
&= a\,\psi\,(t, q_t - i) + b - \varepsilon t \\
&= \xi(t,i)
\end{aligned}$$

where we have the inductive hypothesis in the second inequality, Lemma 4.4.4 in the third inequality and the fourth property of the ψ function in the second to last inequality. □

We are now ready to bound the value obtained by the algorithm in the last phase and thus prove Theorem 4.4.3.

Proof of Theorem 4.4.3:

Consider the end of phase $T-1$. Observe that the nodes $\sigma_{T-1}(k) \in U_i$ and $\sigma_{T-1}(k) \in V_i$ for $k \in \{1, \ldots, \lceil q_{T-1} \rceil\}$ form a vertex cover of the final matching $\mathcal{M}_i^{(T)}$. Because of the degree constraints, this means that the value the algorithm can gain on the last phase T is at most twice the following expression:

$$\begin{aligned}
&\sum_{k=1}^{\lceil q_{T-1} \rceil} \Big(1 - \ell(T-1,k)\Big) \leq \sum_{k=1}^{\lceil q_{T-1} \rceil} \Big(1 - \xi(T-1,k)\Big) \\
&= a \sum_{k=1}^{\lceil q_{T-1} \rceil} \Big(1 - \ \psi(T-1, q_{T-1} - k)\Big) + \varepsilon(T-1)\lceil q_{T-1} \rceil \\
&\leq \varepsilon\, T^2 + a \sum_{y=-1}^{\infty} \Big(1 - \ \psi\left(T-1, \frac{y}{2}\right)\Big) \leq \varepsilon\, T^2 + a\left(2 + \frac{1}{2}\sqrt{T-1}\right) \\
&= O(\sqrt{T}) + \varepsilon\, O(T^2).
\end{aligned}$$

The first equality uses the relation $1 - b = a$, the second inequality is due to $q_{T-1} = (r_{T-1}+1)/2 \leq T/2$ and the change of index $y := 2(q_{T-1} - k)$, while the last inequality is by (4.4.2) and $\xi(t,-1) \leq 1$ for all $t \geq 1$. □

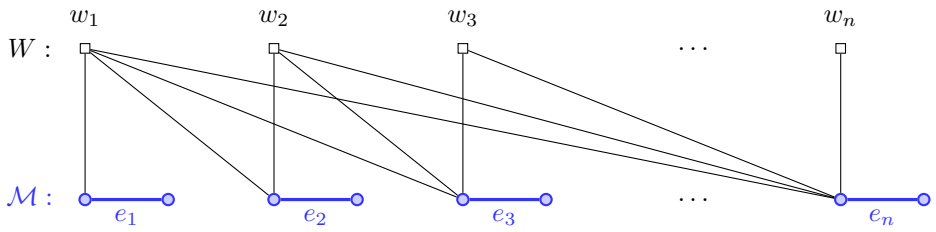


Figure 4.6: An illustration of the instance constructed in Lemma 4.4.6

4.4.5 Connecting the matching $\mathcal{M}^{(t)}$ to the online nodes

In this section, we connect the matching $\mathcal{M}^{(t)} = \cup_{i=1}^{m} \mathcal{M}_i^{(t)}$ to the online vertices to form hyperedges, for every phase $t \in [T]$. The way in which they are connected is similar to the vertex-arrival instance of [KVV90] for bipartite graphs. The main idea is to obfuscate the partners of the online nodes in the optimal matching.

The following construction is an adaptation of the vertex-arrival instance for bipartite graphs [KVV90] to tripartite hypergraphs. Given a graph matching $\mathcal{M}$ on the offline nodes, the first online node connects to every edge in $\mathcal{M}$. After the algorithm sets fractional values on every edge of $\mathcal{M}$, the second online node connects to $\mathcal{M} \setminus \{e_1\}$, where e_1 is the edge in the matching with the lowest fractional value. More generally, for every $k \in \{1, \ldots, |\mathcal{M}|\}$, the k^{th} online node connects to the $|\mathcal{M}| - k + 1$ edges $\mathcal{M} \setminus \{e_1, \ldots, e_{k-1}\}$, and e_k is defined as the edge having the lowest fractional value among them at the end of the k^{th} iteration. This instance is illustrated in Figure 4.6.

We now state the guarantee obtained by this construction, parametrized by the maximum (fractional) degree $\Delta \in [0, 1]$ attained by an offline node.

Lemma 4.4.6. *For any graph matching $\mathcal{M} = (V, E)$, there exists an online tripartite hypergraph instance $\mathcal{H} = (V, W, H)$ such that $\Gamma(\mathcal{H}) = \mathcal{M}$ and $\mathsf{OPT}(\mathcal{H}) = |\mathcal{M}|$. Moreover, for any fractional algorithm $\mathcal{A}$ whose returned solution x satisfies $x(\delta(v)) \leq \Delta$ for all offline nodes $v \in V$, we have*

$$\mathcal{V}(\mathcal{A}, \mathcal{H}) \leq (1 - e^{-\Delta})|\mathcal{M}| + 3/2.$$

Proof:
Let $n := |\mathcal{M}|$. By construction of the instance, observe that $x(e_1) \leq 1/n$. More generally, it is easy to see that:

$$\sum_{k=1}^{\ell} x(e_k) \leq \sum_{k=1}^{\ell} \sum_{i=1}^{k} \frac{1}{n - i + 1} \qquad \forall \ell \in \{1, \ldots, n\}. \tag{4.4.3}$$

The inner sum in (4.4.3) reaches Δ approximately when $k \approx (1 - e^{-\Delta})n$. For higher values of k, it is thus better to use the bound $x(e_k) \leq \Delta$, which holds by

assumption. By defining $p := \lfloor e^{-\Delta} n \rfloor$ and $q := n - p$, we can now compute a precise upper bound on the total value generated by the algorithm using (4.4.3):

$$\begin{aligned}\mathcal{V}(A,\mathcal{H}) = \sum_{k=1}^{n} x(e_k) &\leq \sum_{k=1}^{q}\sum_{i=1}^{k} \frac{1}{n-i+1} + \sum_{k=q+1}^{n} \Delta = \sum_{i=1}^{q}\sum_{k=i}^{q} \frac{1}{n-i+1} + p\Delta \\ &= \sum_{i=1}^{q} \frac{q-i+1}{n-i+1} + p\Delta = q - (n-q)\sum_{i=1}^{q} \frac{1}{n-i+1} + p\Delta \\ &= p\Delta + (n-p) - p\sum_{i=n-q+1}^{n} \frac{1}{i} = p\Delta + n - p - p(H_n - H_p) \qquad (4.4.4)\end{aligned}$$

In order to get the desired result for every value of $n \geq 1$, we now need to tightly approximate the difference of the harmonic numbers $H_n - H_p$. In particular, the well known bounds $\ln(n) + 1/n \leq H_n \leq \ln(n+1)$ for every $n \in \mathbb{N}$ are not enough in this case. We use the equality

$$H_n = \ln(n) + \gamma + \epsilon(n) \qquad \text{for some } 0 < \epsilon(n) < \frac{1}{2n} \qquad (4.4.5)$$

where $\gamma = \lim_{n\to\infty}(H_n - \ln(n)) \approx 0.58$ is Euler's constant. Moreover, recall that

$$e^{-\Delta} n - 1 \leq p \leq e^{-\Delta} n. \qquad (4.4.6)$$

Using (4.4.5) and (4.4.6) together gives:

$$H_n - H_p = \ln\left(\frac{n}{p}\right) + \epsilon(n) - \epsilon(p) \geq \ln\left(\frac{n}{e^{-\Delta}n}\right) - \frac{1}{2p} = \Delta - \frac{1}{2p} \qquad (4.4.7)$$

Finally, plugging (4.4.6) and (4.4.7) into (4.4.4) gets us the desired result for every value of $n \in \mathbb{N}$:

$$\mathcal{V}(A,\mathcal{H}) \leq p\Delta + n - p - p\left(\Delta - \frac{1}{2p}\right) = n - p + \frac{1}{2} \leq (1 - e^{-\Delta})n + \frac{3}{2}.$$

□

The way now in which we apply this construction is by partitioning the matching $\mathcal{M}^{(t)}$ into submatchings based on the load (or fractional degree) of the vertices, and applying Lemma 4.4.6 on each submatching separately. More precisely, let us fix $\eta := |\mathcal{M}^{(t)}|^{-1/3}$ and $N := \lceil 2/\eta \rceil$. We partition the edges of the matching $\mathcal{M}^{(t)}$ into N^2 submatchings as follows

$$\mathcal{M}^{(t)}(i,j) := \left\{ (u,v) \in \mathcal{M}^{(t)} : \ell_u \in \left[\frac{i-1}{N}, \frac{i}{N}\right] \ , \ \ell_v \in \left[\frac{j-1}{N}, \frac{j}{N}\right] \right\}$$

for all $i, j \in [N]$. Then, we apply the construction illustrated in Figure 4.6 to each submatching $\mathcal{M}^{(t)}(i,j)$. This finishes the description of our instance.

4.4.6 Bound for the first $T-1$ phases

Recall from Section 4.4.1 that our instance consists of T phases. For ease of analysis, we will split the total value gained by the algorithm into the value gained in each phase. For an algorithm $\mathcal{A}$, let $\mathcal{V}^{(t)}(\mathcal{A})$ denote the value obtained by $\mathcal{A}$ in phase t. The next lemma upper bounds $\mathcal{V}^{(t)}(\mathcal{A})$ for a threshold-respecting algorithm $\mathcal{A}$, in terms of the loads of the offline nodes at the end of phase $t-1$. Recall that $W^{(t)}$ is the set of online nodes which arrive during phase t.

Lemma 4.4.7. *If $\mathcal{A}$ is threshold-respecting on $W^{(t)}$, then*

$$\mathcal{V}^{(t)}(\mathcal{A}) \leq \sum_{(u,v)\in\mathcal{M}^{(t)}} \Big(1 - f(\ell_u^{(t-1)}) - f(\ell_v^{(t-1)})\Big) + 15\,|\mathcal{M}^{(t)}|^{2/3}.$$

Proof:
Since we are considering a threshold-respecting algorithm, we can compute an upper bound on the amount that the algorithm can put on an edge $e \in \mathcal{M}^{(t)}(i,j)$ while staying below the threshold:

$$\begin{aligned}\Delta(i,j) &:= \max\left\{x : f\left(\frac{i-1}{N} + x\right) + f\left(\frac{j-1}{N} + x\right) \leq 1\right\} \\ &= \ln\left(\frac{e+1}{\exp\left(\frac{i-1}{N}\right) + \exp\left(\frac{j-1}{N}\right)}\right).\end{aligned}$$

A simpler way to write this equation is as follows:

$$\exp(-\Delta(i,j)) = f\left(\frac{i-1}{N}\right) + f\left(\frac{j-1}{N}\right).$$

By Lemma 4.4.6, we know that there exists sets of online nodes $W^{(t)}(i,j)$ which, together with the matchings $\mathcal{M}^{(t)}(i,j)$, form online hypergraphs $\mathcal{H}^{(t)}(i,j)$ such that

$$\mathcal{V}\Big(\mathcal{A}, \mathcal{H}^{(t)}(i,j)\Big) \leq \Big(1 - \exp(-\Delta(i,j))\Big)\left|\mathcal{M}^{(t)}(i,j)\right| + \frac{3}{2} \qquad \forall i,j \in [N].$$

Now, observe that for an edge $\{u,v\} \in \mathcal{M}^{(t)}(i,j)$ with loads $\ell_u^{(t-1)}$ and $\ell_v^{(t-1)}$, we have

$$f\left(\frac{i-1}{N}\right) \geq f\left(\ell_u^{(t-1)} - \frac{1}{N}\right) \geq f(\ell_u^{(t-1)}) - \frac{1}{N}.$$

The first inequality follows from the fact that f is a non-decreasing function. The second inequality follows from the fact that $f'(x) = f(x) \leq 1$ for every $x \in [0,1]$. Similarly, we have

$$f\left(\frac{j-1}{N}\right) \geq f(\ell_v^{(t-1)}) - \frac{1}{N}.$$

Hence, the value gained by the algorithm in phase t can be upper bounded as

$$\begin{aligned}
\mathcal{V}^{(t)}(\mathcal{A}) &= \sum_{i,j=1}^{N} \mathcal{V}\Big(\mathcal{A}, \mathcal{H}^{(t)}(i,j)\Big) \\
&\leq \sum_{i,j=1}^{N} \left(1 - f\left(\frac{i-1}{N}\right) - f\left(\frac{j-1}{N}\right)\right) \left|\mathcal{M}^{(t)}(i,j)\right| + \frac{3}{2}N^2 \\
&\leq \sum_{i,j=1}^{N} \sum_{(u,v)\in\mathcal{M}^{(t)}(i,j)} \left(1 - f(\ell_u^{(t-1)}) - f(\ell_v^{(t-1)}) + \frac{2}{N}\right) + \frac{3}{2}N^2 \\
&= \sum_{(u,v)\in\mathcal{M}^{(t)}} \big(1 - f(\ell_u^{(t-1)}) - f(\ell_v^{(t-1)})\big) + \frac{2}{N}|\mathcal{M}^{(t)}| + \frac{3}{2}N^2 \\
&\leq \sum_{(u,v)\in\mathcal{M}^{(t)}} \big(1 - f(\ell_u^{(t-1)}) - f(\ell_v^{(t-1)})\big) + \eta|\mathcal{M}^{(t)}| + 14\eta^{-2}.
\end{aligned}$$

For the last inequality, since $N = \lceil 2/\eta \rceil$, we have used the bounds $2/N \leq \eta$ and $N^2 \leq (2/\eta + 1)^2 = (4 + 4\eta + \eta^2)/\eta^2 \leq 9/\eta^2$ because $\eta \in (0, 1]$. In fact, $\eta = |\mathcal{M}^{(t)}|^{-1/3}$ so the bound becomes

$$\mathcal{V}^{(t)}(\mathcal{A}) \leq \sum_{(u,v)\in\mathcal{M}^{(t)}} \big(1 - f(\ell_u^{(t-1)}) - f(\ell_v^{(t-1)})\big) + 15|\mathcal{M}^{(t)}|^{2/3}.$$

□

From the definition of f and the construction of the matching $\mathcal{M}^{(t)}$, we can convert the previous bound into the following expression. We remark that the threshold-respecting property is only used in the proof of Lemma 4.4.7.

Lemma 4.4.8. *If $\mathcal{A}$ is threshold-respecting on $W^{(t)}$, then*

$$\mathcal{V}^{(t)}(\mathcal{A}) \leq \frac{e-1}{e+1}\, m + 15\, t^{2/3} m^{2/3}.$$

Proof:
By splitting the matching $\mathcal{M}^{(t)}$ based on the m components, we can rewrite the bound in Lemma 4.4.7 as

$$\mathcal{V}^{(t)}(\mathcal{A}) \leq \sum_{i=1}^{m} \sum_{(u,v)\in\mathcal{M}_i^{(t)}} \big(1 - f(\ell_u^{(t-1)}) - f(\ell_v^{(t-1)})\big) + 15\,|\mathcal{M}^{(t)}|^{2/3}. \tag{4.4.8}$$

Fix a component $i \in [m]$, and let $\mathcal{E}_i := \Big\{e \in \mathcal{M}_i^{(t-1)} \mid f(\ell_u^{(t-1)}) + f(\ell_v^{(t-1)}) \geq 1\Big\}$ be the subset of edges in the matching $\mathcal{M}_i^{(t-1)}$ which exceed the threshold at the

end of phase $t-1$. By the construction of $\mathcal{M}_i^{(t)}$ in Section 4.4.3, we know that its node set consists of the nodes incident to $\mathcal{E}_i$, in addition to two new fresh nodes whose load is 0 at the end of phase $t-1$. This allows us to expand the inner sum in (4.4.8) as:

$$\sum_{(u,v)\in\mathcal{M}_i^{(t)}} 1 - f(\ell_u^{(t-1)}) - f(\ell_v^{(t-1)}) = 1 - 2f(0) + \sum_{(u,v)\in\mathcal{E}_i} 1 - f(\ell_u^{(t-1)}) - f(\ell_v^{(t-1)})$$
$$\leq 1 - 2f(0) = \frac{e-1}{e+1}$$

where the inequality follows from definition of $\mathcal{E}_i$. Plugging this into (4.4.8) with the bound $|\mathcal{M}^{(t)}| \leq tm$ (which is immediate by the construction in Section 4.4.3) yields the desired result. □

For an ε-threshold-respecting algorithm, we pick up an extra εtm term.

Corollary 4.4.9. *If $\mathcal{A}$ is ε-threshold-respecting on $W^{(t)}$ for some $\varepsilon \geq 0$, then*

$$\mathcal{V}^{(t)}(A) \leq \frac{e-1}{e+1}m + 15\,(tm)^{2/3} + \varepsilon tm.$$

Proof:
Fix an edge $e \in \mathcal{M}^{(t)}$. Let $h_1, h_2, \ldots, h_k$ be the hyperedges arriving in phase t which contain e, denoted such that h_i arrives before h_j if and only if $i < j$. Let $j \in [k]$ be the smallest index such that $\phi(h_j) > 1$ immediately after $\mathcal{A}$ assigns x_{h_j} to h_j. Let $z_{h_j} \geq 0$ be the largest value such that $\phi(h_j) \geq 1$ if $\mathcal{A}$ were to assign $x_{h_j} - z_{h_j}$ to h_j instead. Define $z_{h_i} := 0$ for all $i < j$, and $z_{h_i} := x_{h_i}$ for all $i > j$. Since $\mathcal{A}$ is ε-threshold-respecting on $W^{(t)}$, we have $\sum_{i=1}^{k} z_{h_i} \leq \varepsilon$ because f is convex and $f' = f$.

Let z be the vector obtained by repeating this procedure on every edge $e \in \mathcal{M}^{(t)}$. Then, $\mathbb{1}^\top z \leq \varepsilon tm$ as $|\mathcal{M}^{(t)}| \leq tm$. Moreover, observe that the algorithm which assigns $x - z$ in phase t is threshold-respecting on $W^{(t)}$. Thus, we can apply Lemma 4.4.8 to obtain the desired upper bound on $\mathbb{1}^\top(x - z)$. □

4.4.7 Putting everything together

In this section, we complete the proof of Theorem 4.1.1. Since we assumed that the algorithm is ε-threshold respecting in the first $T-1$ phases, we can apply Corollary 4.4.9 to upper bound the value obtained in the first $T-1$ phases as

$$\sum_{t=1}^{T-1}\left(\frac{e-1}{e+1}m + \varepsilon tm + O\left((tm)^{2/3}\right)\right) \leq \frac{e-1}{e+1}Tm + \varepsilon T^2 m + O\left(T^{5/3}m^{2/3}\right).$$

By Theorem 4.4.3, the value gained by the algorithm on each component C_i during the last phase T is at most $O(\sqrt{T} + \varepsilon T^2)$. Hence, the algorithm gains at most $O(\sqrt{T}m + \varepsilon T^2 m)$ in the last phase.

We now argue that our instance $\mathcal{H} = (V, W, H)$ has a perfect matching.

Lemma 4.4.10. *Our adversarial instance $\mathcal{H} = (V, W, H)$ satisfies*

$$\mathsf{OPT}(\mathcal{H}) = Tm.$$

Proof:
We prove that for every $t \in [T]$, there exists a hypergraph matching of size tm at the end of phase t. Let C_i be a component with bipartition $U_i = [T]$ and $V_i = [T]$. It suffices to show that there exists a graph matching $\widetilde{\mathcal{M}}_i^{(t)}$ with vertex set $[t]$ on each side. This is because $\widetilde{\mathcal{M}}^{(t)} := \cup_{i=1}^{m} \widetilde{\mathcal{M}}_i^{(t)}$ can be extended to a hypergraph matching in $\mathcal{H}$ by our construction (see Lemma 4.4.6). Let $E_i^{(t)} \subseteq \mathcal{M}_i^{(t)}$ be the edges whose endpoints are not active at the end of phase t. Then, a simple inductive argument on $t \geq 1$ shows that $\cup_{s=1}^{t-1} E_i^{(s)} \cup \mathcal{M}_i^{(t)}$ is a graph matching with vertex set $[t]$ on each side (see Figure 4.3 for an example). □

By Lemma 4.4.10, the competitive ratio of the algorithm is at most

$$\frac{e-1}{e+1} + O(\varepsilon T + T^{2/3} m^{-1/3} + T^{-1/2}).$$

Hence, letting $m \to \infty$, picking $T = o(\sqrt{m})$ such that $T \to \infty$ and setting $\varepsilon = o(1/T)$, we conclude that the competitive ratio is upper bounded by $(e-1)/(e+1)$, thus finishing the proof of Theorem 4.1.1.

4.5 Integral algorithm for bounded degree hypergraphs

In this section, we show that RANDOM (Algorithm 4.5.1) performs better than the greedy algorithm when the online nodes have bounded degree. We prove Theorem 4.1.2, restated below.

Theorem 4.5.1. *Algorithm 4.5.1 is ρ-competitive for k-uniform hypergraphs whose online nodes have degree at most d, where*

$$\rho = \min\left(\frac{1}{k-1}, \frac{d}{(d-1)k+1}\right).$$

Proof:
Let the algorithm be denoted by $\mathcal{A}$. We prove the result via a primal-dual analysis,

Algorithm 4.5.1 RANDOM algorithm for bounded degree hypergraphs

Input : k-uniform hypergraph $\mathcal{H} = (V, W, H)$ with online arrivals of each $w \in W$ with $|\delta(w)| \leq d$.
Output : Matching $\mathcal{M} \subset H$

set $\mathcal{M} \leftarrow \emptyset$
when $w \in W$ **arrives with** $\delta(w) \subseteq H$**:**
pick uniformly at random $h \in \delta(w)$ among the hyperedges that are disjoint from $\mathcal{M}$
set $y_v = \min\left(\frac{1}{k-1}, \frac{d}{(d-1)k+1}\right)$ for all $v \in h \setminus \{w\}$
set $y_w = \max\left(0, \frac{d-k+1}{(d-1)k+1}\right)$
return $\mathcal{M}$

where the random primal solution is given by $x_h := \mathbb{1}_{\{h \in \mathcal{M}\}}$ for every $h \in H$ and the random dual solution is the vector $y \in [0,1]^{V \cup W}$ constructed during the execution of the algorithm. Observe that the objective values of both solutions are equal at all times during the execution of the algorithm:

$$\mathcal{V}(\mathcal{A}) = |\mathcal{M}| = \sum_{h \in H} x_h = \sum_{v \in V \cup W} y_v. \tag{4.5.1}$$

This holds since every time a hyperedge $h \in H$ is matched by the algorithm, increasing the primal value $\mathcal{V}(\mathcal{A})$ by one, the dual objective increases by $\sum_{v \in h} y_v$. Two easy computations that we omit show that the latter is also equal to one in both cases where $d \leq k-1$ and $d \geq k-1$.

We will now show that, in expectation, the dual constraints are satisfied up to a factor of ρ, i.e.

$$\mathbb{E}\left[\sum_{v \in h} y_v\right] \geq \rho \qquad \forall h \in H. \tag{4.5.2}$$

This will imply the theorem, since the random vector $\mathbb{E}[y]/\rho$ will then be a feasible dual solution, leading to $\mathbb{E}[\mathcal{V}(\mathcal{A})] = \mathbb{E}\left[\sum_{v \in V \cup W} y_v\right] \geq \rho\, \mathsf{OPT}_{\mathsf{LP}}$ by (4.5.1) and (4.5.2).

To show this inequality, let $h \in H$ be an arbitrary hyperedge incident to some online node $w \in W$. We now consider the following probabilistic event upon the arrival of w:

$$\mathcal{E} := \Big\{\exists v \in h \setminus \{w\} \text{ which is already matched at the arrival of } w\Big\}.$$

We will show (4.5.2) by conditioning on $\mathcal{E}$ and on its complementary event $\bar{\mathcal{E}}$, which states that all nodes in $h \setminus \{w\}$ are unmatched when w arrives, and that the hyperedge h is thus available and considered in the random choice of the

algorithm in this step. In the first case, if $\mathcal{E}$ happens, then some offline node $u \in h \setminus \{w\}$ has already had its dual value set to $y_u = \min\left(\frac{1}{k-1}, \frac{d}{(d-1)k+1}\right) = \rho$ in a previous step of the algorithm, leading to

$$\mathbb{E}\left[\sum_{v \in h} y_v \,\middle|\, \mathcal{E}\right] = \sum_{v \in h} \mathbb{E}\left[y_v \mid \mathcal{E}\right] \geq \mathbb{E}\big[y_u \mid \mathcal{E}\big] = \rho.$$

Otherwise, if $\bar{\mathcal{E}}$ happens, we know that with probability at least $1/d$, the algorithm adds h to the matching. Summing the dual values of the offline nodes contained in h gives

$$\sum_{v \in h \setminus \{w\}} \mathbb{E}\left[y_v \mid \bar{\mathcal{E}}\right] \geq \frac{1}{d} \cdot (k-1) \cdot \min\left(\frac{1}{k-1}, \frac{d}{(d-1)k+1}\right).$$

Furthermore, since the algorithm will always match w to a hyperedge in this case, we have:

$$\mathbb{E}\left[y_w \mid \bar{\mathcal{E}}\right] = \max\left(0, \frac{d-k+1}{(d-1)k+1}\right).$$

Adding those terms together, we get:

$$\begin{aligned}
\sum_{v \in h} \mathbb{E}\left[y_v \mid \bar{\mathcal{E}}\right] &\geq \frac{1}{d} \cdot (k-1) \cdot \min\left(\frac{1}{k-1}, \frac{d}{(d-1)k+1}\right) + \max\left(0, \frac{d-k+1}{(d-1)k+1}\right) \\
&= \min\left(\frac{1}{d}, \frac{k-1}{(d-1)k+1}\right) + \max\left(0, \frac{d-(k-1)}{(d-1)k+1}\right) \\
&\geq \min\left(\frac{1}{d}, \frac{k-1}{(d-1)k+1} + \frac{d-(k-1)}{(d-1)k+1}\right) \geq \rho.
\end{aligned}$$

This shows that (4.5.2) holds, and hence proves that the algorithm is ρ-competitive. □

4.6 Justification of assumptions in Section 4.4.2

In this section, we justify the two assumptions made on the algorithm in Section 4.4.2.

4.6.1 Assumption 1: Symmetry

We start by justifying the symmetry assumption. For a vertex-arrival hypergraph $\mathcal{H}$, we denote $V(\mathcal{H})$ as the set of offline nodes, $W(\mathcal{H})$ as the set of online nodes, and $H(\mathcal{H})$ as the set of hyperedges.

Definition 4.6.1. Given a vertex-arrival hypergraph $\mathcal{H} = (V, W, H)$, an *automorphism* is a permutation σ of the offline nodes V such that for every $S \subseteq V$ and $w \in W$,

$$S \cup \{w\} \in H \iff \{\sigma(v) : v \in S\} \cup \{w\} \in H.$$

For $S \subseteq V$, we write $\sigma(S) \coloneqq \{\sigma(v) : v \in S\}$ for the sake of brevity. For a hyperedge $h = S \cup \{w\}$ where $S \subseteq V$ and $w \in W$, we denote $\sigma(h) \coloneqq \sigma(S) \cup w$. We also denote the relabelled hypergraph after applying σ to $\mathcal{H}$ as

$$\sigma(\mathcal{H}) \coloneqq (\sigma(V), W, \{\sigma(h) : h \in H\}).$$

Definition 4.6.2. Given a vertex-arrival hypergraph $\mathcal{H} = (V, W, H)$, let Σ be a subset of its automorphisms. We say that $\mathcal{H}$ is Σ-*symmetric*. A fractional matching x in $\mathcal{H}$ is Σ-*symmetric* if $x_h = x_{\sigma(h)}$ for all $h \in H$ and $\sigma \in \Sigma$. An algorithm $\mathcal{A}$ is Σ-*symmetric* on $\mathcal{H}$ if it outputs a Σ-symmetric fractional matching given $\mathcal{H}$.

Since the construction of our vertex-arrival instance depends on the behaviour of the algorithm, we will overload the notation $\mathcal{H}$ as follows. An *(adaptive vertex-arrival) instance* is a function $\mathcal{H}$ which takes as input an algorithm $\mathcal{A}$ and outputs a vertex-arrival hypergraph $\mathcal{H}(\mathcal{A})$. For $i \geq 1$, let $\mathcal{H}_i(\mathcal{A})$ be the subgraph of $\mathcal{H}(\mathcal{A})$ right after the arrival of the ith online node w_i. We assume that $\mathcal{H}_1(\mathcal{A}) = \mathcal{H}_1(\mathcal{A}')$ for any pair of algorithms $\mathcal{A}$ and $\mathcal{A}'$. Note that this implies $V(\mathcal{H}(\mathcal{A})) = V(\mathcal{H}(\mathcal{A}'))$.

Definition 4.6.3. Let Σ be a subgroup of permutations of V such that $\mathcal{H}(\mathcal{A})$ is Σ-symmetric for every Σ-symmetric algorithm $\mathcal{A}$. Given an algorithm $\mathcal{A}$, we define the Σ-*symmetrization* of $\mathcal{A}$, denoted $\mathrm{sym}_\Sigma(\mathcal{A})$, as the algorithm that sets:

$$x_h \coloneqq \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} x'_{\sigma(h)} \qquad \forall h \in H(\mathcal{H}),$$

where $x'_{\sigma(h)}$ is the value that $\mathcal{A}$ would assign to hyperedge $\sigma(h)$ when running on $\mathcal{H}(\mathrm{sym}_\Sigma(\mathcal{A}))$.

We make a couple of observations about $\mathrm{sym}_\Sigma(\mathcal{A})$. First, $\mathrm{sym}_\Sigma(\mathcal{A})$ is well-defined for any algorithm $\mathcal{A}$. Recall that we assumed $\mathcal{H}_1(\mathcal{A}) = \mathcal{H}_1(\mathcal{A}')$ for any other algorithm $\mathcal{A}'$. In particular, $\mathcal{H}_1(\mathcal{A}) = \mathcal{H}_1(\mathrm{sym}_\Sigma(\mathcal{A}))$. For every $i \geq 1$, when the ith online node w_i arrives in $\mathcal{H}(\mathrm{sym}_\Sigma(\mathcal{A}))$, $\mathcal{A}$ assigns fractional values to the hyperedges in $\delta(w_i)$. This defines how $\mathrm{sym}_\Sigma(\mathcal{A})$ assigns fractional values to the hyperedges in $\delta(w_i)$, which in turn determines what the next set of hyperedges $\delta(w_{i+1})$ will be in $\mathcal{H}(\mathrm{sym}_\Sigma(\mathcal{A}))$.

Next, $\mathrm{sym}_\Sigma(\mathcal{A})$ is Σ-symmetric. For any hyperedge $h \in H(\mathcal{H}(\mathrm{sym}_\Sigma(\mathcal{A})))$ and permutation $\tau \in \Sigma$, we have

$$x_{\tau(h)} = \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} x'_{\sigma(\tau(h))} = \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} x'_{\sigma(h)} = x_h,$$

where the second equality is due to Σ being a group. Note that for a Σ-symmetric algorithm $\mathcal{A}$, we have $\mathrm{sym}_\Sigma(\mathcal{A}) = \mathcal{A}$.

We now show that one only needs to define the behavior of a Σ-symmetric instance $\mathcal{H}$ for a Σ-symmetric algorithm $\mathcal{A}$. That is, any Σ-symmetric instance $\mathcal{H}$ defined for Σ-symmetric algorithms can be extended to a Σ-symmetric instance $\mathcal{H}'$ defined for all algorithms.

Lemma 4.6.4. *Let $\mathcal{H}(\mathcal{A})$ be a Σ-symmetric instance defined for all Σ-symmetric algorithms $\mathcal{A}$. There is an instance $\mathcal{H}'$ defined for all algorithms, such that for any algorithm $\mathcal{A}$:*

- *$\mathcal{H}'(\mathcal{A})$ is Σ-symmetric,*
- *$\mathcal{H}'(\mathcal{A}) = \mathcal{H}(\mathrm{sym}_\Sigma(\mathcal{A}))$,*
- *$\mathcal{V}(\mathcal{A}, \mathcal{H}') = \mathcal{V}(\mathrm{sym}_\Sigma(\mathcal{A}), \mathcal{H})$.*

Proof:
We define $\mathcal{H}'(\mathcal{A}) := \mathcal{H}(\mathrm{sym}_\Sigma(\mathcal{A}))$ for any algorithm $\mathcal{A}$. This immediately gives the first two properties. For the last property, let x be the output of $\mathrm{sym}_\Sigma(\mathcal{A})$ on $\mathcal{H}(\mathrm{sym}_\Sigma(\mathcal{A}))$ and let x' be the output of $\mathcal{A}$ on $\mathcal{H}'(\mathcal{A}) = \mathcal{H}(\mathrm{sym}_\Sigma(\mathcal{A}))$. By definition, we have $x_h := \frac{1}{|\Sigma|}\sum_{\sigma\in\Sigma} x'_{\sigma(h)}$ for every hyperedge $h \in H(\mathcal{H}(\mathrm{sym}_\Sigma(\mathcal{A})))$. Hence,

$$\sum_h x_h = \sum_h \frac{1}{|\Sigma|}\sum_{\sigma\in\Sigma} x'_{\sigma(h)} = \frac{1}{|\Sigma|}\sum_{\sigma\in\Sigma}\sum_h x'_{\sigma(h)} = \sum_h x'_h.$$

□

In Section 4.4.2, we assumed that the algorithm treats the kth vertex in U_i, say $u_{i,k}$, and the kth vertex in V_i, say $v_{i,k}$, symmetrically. If our constructed hypergraph $\mathcal{H}$ was symmetric with respect to these vertices, i.e. if the permutation σ swapping $u_{i,k}$ and $v_{i,k}$ for all i and k was an automorphism of $\mathcal{H}$, then Lemma 4.6.4 would show that this assumption can be made without loss of generality. In particular, using the subgroup $\Sigma = (\{\sigma, e\}, \circ)$ where e is the identity permutation, it shows that $\mathcal{H}$ can be extended to all algorithms $\mathcal{A}$ so that $\mathcal{A}$ and $\mathrm{sym}_\Sigma(\mathcal{A})$ have the same performance.

However, one part of the instance that breaks this symmetry is the construction given in the proof of Lemma 4.4.6 and illustrated in Figure 4.6. As a reminder, this construction is repeatedly applied to submatchings of $\mathcal{M}^{(t)}$ in Section 4.4.5. Let us fix one such submatching and denote it by $\mathcal{M} := \mathcal{M}^{(t)}(i, j)$. As described in Section 4.4.3 and illustrated in Figure 4.3, if some $u_{i,k} \in \mathcal{M}$, then $v_{i,k} \in \mathcal{M}$ and the submatching is symmetric with respect to this pair, i.e. $\sigma(e) \in \mathcal{M}$ for every edge e in $\mathcal{M}$. However, due to the Lemma 4.4.6 construction, $e \cup \{w\}$ might be a hyperedge in $\mathcal{H}$ for some online vertex w, while $\sigma(e \cup \{w\}) = \sigma(e) \cup \{w\}$ might not be a hyperedge in $\mathcal{H}$.

To fix this, the construction can be slightly tweaked in the following way. An important observation is that the horizontal edges in $\mathcal{M}$ (between $u_{i,k}$ and $v_{i,k}$) are not isomorphic to any other edge in the hypergraph, whereas each of the diagonal edges (non-horizontal edges) are isomorphic to exactly one other edge in $\mathcal{M}$. For this reason, we can first apply the Lemma 4.4.6 construction on just the horizontal edges of $\mathcal{M}$.

We can then apply a slightly modified construction to the diagonal edges, where the pairs of isomorphic edges are treated in the same way. In the original construction, a newly arriving online vertex w would be connected to all edges in $\mathcal{M}$ that were incident to the previous online vertex, except for the one with the smallest fractional value. In the modified construction, we instead consider the online vertices in groups of two. For every two consecutive online vertices, we connect them to all edges in $\mathcal{M}$ that were incident to the previous online vertex, except for the diagonal pair with the smallest total fractional value. This ensures that the symmetry between the diagonal edges is respected.

This modified construction would slightly worsen the upper bound in Lemma 4.4.6. Let $\mathcal{M}_{\text{hor}}$ be the set of horizontal edges in $\mathcal{M}$ and $\mathcal{M}_{\text{diag}}$ be the set of diagonal edges in $\mathcal{M}$. By applying Lemma 4.4.6 to the horizontal edges, we get that the value of the matching is at most $(1 - e^{-\Delta})|\mathcal{M}_{\text{hor}}| + 3/2$. The value of the diagonal edges is at most twice the value of the original construction from Lemma 4.4.6 applied to a transversal of the pairs of diagonal edges, which is at most $(1-e^{-\Delta}) \cdot \frac{1}{2}|\mathcal{M}_{\text{diag}}|+3/2$ by Lemma 4.4.6. So the total value of the matching is at most:

$$(1 - e^{-\Delta})|\mathcal{M}_{\text{hor}}| + 3/2 + 2 \cdot \left((1 - e^{-\Delta}) \cdot \frac{1}{2}|\mathcal{M}_{\text{diag}}| + 3/2\right) = (1 - e^{-\Delta})|\mathcal{M}| + 9/2.$$

This results in a constant of $9/2$ instead of $3/2$ in Lemma 4.4.6, and a constant of 42 instead of 15 in Lemma 4.4.7, Lemma 4.4.8 and Corollary 4.4.9. This does not affect the asymptotic upper bound for large m. Hence, it shows that Theorem 4.1.1 also holds for non-symmetric algorithms.

4.6.2 Assumption 2: There is an optimal ε-threshold respecting algorithm

Next, we justify that we restrict to ε-threshold respecting algorithms in our proof. Let $\mathcal{H}$ be the instance constructed in Section 4.4.1. Let f be the function given by

$$f(x) \coloneqq \frac{e^x}{e+1},$$

and recall the definition of ε-threshold respecting with respect to f (Definition 4.4.1).

For any algorithm $\mathcal{A}$ and $\varepsilon > 0$, we now show that there exists an algorithm $\mathcal{A}'$ which is ε-threshold respecting on all online nodes before the last phase. Moreover, there exists an instance $\mathcal{H}'$ such that the performance of $\mathcal{A}$ on $\mathcal{H}'$ matches the performance of $\mathcal{A}'$ on $\mathcal{H}$.

Lemma 4.6.5. *Let $\mathcal{H}$ be the instance constructed in Section 4.4.1. For any algorithm $\mathcal{A}$ and $\varepsilon > 0$, there exists an algorithm $\mathcal{A}'$ which is ε-threshold respecting on all online nodes before the last phase. Furthermore, there exists an instance $\mathcal{H}'$ such that*

$$\frac{\mathcal{V}(\mathcal{A}, \mathcal{H}'(\mathcal{A}))}{\mathsf{OPT}(\mathcal{H}'(\mathcal{A}))} = \frac{\mathcal{V}(\mathcal{A}', \mathcal{H}(\mathcal{A}'))}{\mathsf{OPT}(\mathcal{H}(\mathcal{A}'))}.$$

Proof:
From $\mathcal{H}$, we construct a new instance $\mathcal{H}'$ as follows. Let $N = \lceil 2/\varepsilon \rceil$. For every offline node v in $\mathcal{H}$, create N offline copies in $\mathcal{H}'$, denoted $v'_1, v'_2, \ldots, v'_N$. The new algorithm $\mathcal{A}'$ will be defined based on the behaviour of $\mathcal{A}$ on $\mathcal{H}'$. When the ith online node w_i arrives in $\mathcal{H}(\mathcal{A}')$ for $i \geq 1$, at most N copies of w_i arrives sequentially in $\mathcal{H}'(\mathcal{A})$, denoted $w'_{i,1}, w'_{i,2}, \ldots$. When the jth copy $w'_{i,j}$ arrives, for every edge $h = S \cup w_i$ in $\mathcal{H}_i(\mathcal{A}')$, add the edge $h'_j \coloneqq \{v'_j : v \in S\} \cup w'_{i,j}$ to $\mathcal{H}'(\mathcal{A})$. Now, let $x'_{i,j}$ denote the solution given by $\mathcal{A}$ in $\mathcal{H}'$ after the arrival of $w'_{i,j}$. Consider the following averaged solution

$$x_{i,j}(h) \coloneqq \frac{1}{N} \sum_k x'_{i,j}(h'_k) \qquad \forall h \in H(\mathcal{H}_i(\mathcal{A}')).$$

If $j = N$, or w_i appeared before the last phase and there exists a hyperedge $h \in \delta(w_i)$ such that

$$\sum_{v \in h \setminus \{w_i\}} f(x_{i,j}(\delta(v))) \geq 1,$$

then $w'_{i,j+1}, \ldots, w'_{i,N}$ will not arrive in $\mathcal{H}'$. In this case, $\mathcal{A}'$ sets $x(h) \leftarrow x_{i,j}(h)$ for all $h \in \delta(w_i)$ in $\mathcal{H}$. Otherwise, we proceed to let the $(j+1)$th copy $w'_{i,j+1}$ arrive in $\mathcal{H}'$. This completes the description of $\mathcal{A}'$.

Clearly, x is a fractional matching in $\mathcal{H}(\mathcal{A}')$. Moreover,

$$\mathcal{V}(\mathcal{A}', \mathcal{H}(\mathcal{A}')) = \sum_h x(h) = \frac{\mathcal{V}(\mathcal{A}, \mathcal{H}'(\mathcal{A}))}{N}.$$

Next, we claim that $N \cdot \mathsf{OPT}(\mathcal{H}(\mathcal{A}')) = \mathsf{OPT}(\mathcal{H}'(\mathcal{A}))$. Based on the construction of $\mathcal{H}$, the offline optimal matching in $\mathcal{H}(\mathcal{A}')$ covers all the online nodes in the last phase, and the online nodes on which $\mathcal{A}'$ is strictly threshold respecting. Let W_{OPT} denote the union of these two sets. For each $w_i \in W_{\mathsf{OPT}}$, observe that $w_{i,j}$ is present in $\mathcal{H}'(\mathcal{A})$ for all $j \in [N]$ by our construction of $\mathcal{H}'$. Hence, the offline optimal matching in $\mathcal{H}'(\mathcal{A})$ covers the following online nodes

$$\{w_{i,j} : w_i \in W_{\mathsf{OPT}}, j \in [N]\}.$$

So, $N \cdot \mathsf{OPT}(\mathcal{H}(\mathcal{A}')) = \mathsf{OPT}(\mathcal{H}'(\mathcal{A}))$ as desired.

It is left to show that $\mathcal{A}'$ is ε-threshold respecting on all online nodes before the last phase. Pick such an online node w_i and let $w_{i,j}$ be its last copy in $\mathcal{H}'(\mathcal{A})$. Note that $x_{i,j}$ is the output of $\mathcal{A}'$ in $\mathcal{H}_i(\mathcal{A}')$. For any $h \in \delta(w_i)$, we have

$$\begin{aligned}
\sum_{v \in h \setminus \{w_i\}} f(x_{i,j}(\delta(v))) &\leq \sum_{v \in h \setminus \{w_i\}} f\left(x_{i,j-1}(\delta(v)) + \frac{1}{N}\right) && (x'_{i,j}(\delta(w'_{i,j})) \leq 1) \\
&\leq \sum_{v \in h \setminus \{w_i\}} \left(f(x_{i,j-1}(\delta(v)) + \frac{1}{N}\right) && (f \text{ is 1-Lipschitz}) \\
&< 1 + \frac{2}{N} && (\text{due to } |h| = 3 \text{ and the construction of } \mathcal{H}') \\
&\leq 1 + \varepsilon
\end{aligned}$$

□

4.7 Integral upper bound for k-uniform hypergraphs

In this section, we prove a strong upper bound against any randomized integral algorithm, showing that the greedy algorithm is almost optimal, since it achieves a competitive ratio of $1/k$.

Theorem 4.7.1. *For the online matching problem on k-uniform hypergraphs, no randomized integral algorithm can be $2/k$-competitive.*

Proof:
We prove that any randomized integral algorithm is at most $(2 - 2^{-k+1})/k$-competitive. To do so, we make use of Yao's principle [Yao77]: it suffices to construct a randomized instance for which any deterministic integral algorithm is at most $(2 - 2^{-k+1})/k$-competitive in expectation. Let us now describe our randomized construction $\mathcal{H} = (V, W, H)$ for any $k \in \mathbb{N}$.

- The offline nodes are partitioned into $k-1$ blocks: $V = C_1 \cup \cdots \cup C_{k-1}$, where $|C_i| = 2(k-i)$ for each $i \in \{1, \ldots, k-1\}$, meaning that $|V| = k(k-1)$.
- The instance first consists of $k-1$ phases with online nodes $w_1, \ldots, w_{k-1}$ arriving, all of which are incident to 2 hyperedges. For every $i \in \{1, \ldots, k-1\}$, both hyperedges incident to w_i are disjoint on the offline nodes V and they will both contain $(k-i)$ nodes from C_i, as well as 1 node from each C_j for $j \in \{1, \ldots, i-1\}$.

- We now construct a random subset of hyperedges $H_1 \subseteq H$ in the following way. At the arrival of w_i for every $i \leq k-1$, pick one of the two incident hyperedges uniformly at random and put it in H_1. We denote by $V(H_1)$ the offline nodes spanned by H_1.

- Our construction will now satisfy the following property. For every $i \in \{1, \dots, k-1\}$, after the arrival of w_i and the random choice described above, we have that

$$|C_j \setminus V(H_1)| = k - i \qquad \forall j \in \{1, \dots, i\}. \tag{4.7.1}$$

  After a certain phase $i - 1 \leq k - 2$, the two hyperedges incident to w_i in the next iteration are then both constructed as follows: take $k - i$ nodes from C_i and complete it by arbitrarily picking one node from $C_j \setminus V(H_1)$ for every $j \leq i - 1$.

- After phase $k - 1$, we have that $|C_j \setminus V(H_1)| = 1$ for every $j \leq k - 1$, by invariant (4.7.1). The instance now makes one more online node w_k incident to one hyperedge arrive, whose offline nodes are $C_j \setminus V(H_1)$ for every $j \in \{1, \dots, k-1\}$. Let us also add this hyperedge to H_1.

Let us first show that (4.7.1) holds by induction. In the first phase, both hyperedges partition C_1 on the offline nodes, since $|C_1| = 2(k-1)$. One of them is chosen to enter H_1, meaning that $C_1 \setminus V(H_1) = k - 1$ after phase 1. Let us now fix a phase $i \leq k - 1$ and suppose that (4.7.1) holds for all previous phases. By construction, C_i is completely covered by the two hyperedges arriving at phase i, since $|C_i| = 2(k - i)$ and both of these hyperedges contain $k - i$ nodes from C_i. One of these hyperedges enters H_1 at the end of phase i, meaning that $|C_i \setminus V(H_1)| = k - i$ indeed holds. For any other C_j with $j < i$, note that $|C_j \setminus V(H_1)| = k - i + 1$ at the beginning of phase i, by induction hypothesis. Both hyperedges coming at phase i intersect C_j at two different nodes, one of which enters $V(H_1)$ by the random choice, meaning that $|C_j \setminus V(H_1)|$ drops by 1 and equals $k - i$, thus showing (4.7.1).

Observe that, by construction, the hyperedges in H_1 are all disjoint from each other. Since we add one hyperedge to H_1 for every online node, we get that $\mathsf{OPT}(\mathcal{H}) = k$.

Let us now upper bound the value that any deterministic algorithm can get on this randomized instance. The key observation is that, if the algorithm picks a hyperedge $h \in \delta(w_i)$ which is not placed in H_1 for some phase $i \in \{1, \dots, k-1\}$, then it cannot pick any hyperedge arriving in later iterations. This holds, since in that case, $C_i \setminus V(H_1) \subseteq h$, and any hyperedges arriving in later iterations necessarily intersect $C_i \setminus V(H_1)$ by construction.

Let us denote by $\mathcal{V}_i$ the maximum expected value that a deterministic algorithm can get if we were to start the instance from phase i and go up to phase k.

Clearly, $\mathcal{V}_k = 1$. For a phase $i \in \{1, \dots, k-1\}$, the algorithm can either choose not to select anything, or it picks a hyperedge and cannot pick anything in later iterations with probability 1/2. We thus get the following recurrence relation:

$$\mathcal{V}_i = \max\left\{\mathcal{V}_{i+1}, \frac{1}{2} + \frac{1}{2}(1 + \mathcal{V}_{i+1})\right\} = \max\left\{\mathcal{V}_{i+1}, 1 + \frac{1}{2}\mathcal{V}_{i+1}\right\}.$$

It is easily checked that the solution to this recurrence is a geometric series $\mathcal{V}_{k-i} = \sum_{j=0}^{i} 2^{-j}$ and thus $\mathcal{V}_1 = \sum_{j=0}^{k-1} 2^{-j} = 2 - 2^{-k+1}$. We have therefore just shown that any algorithm is at most $(2 - 2^{-k+1})/k$ competitive. □

4.8 Rounding algorithm for online hypergraph b-matching

In this section, we consider the online b-matching problem on k-uniform hypergraphs, in which every (offline and online) node v can be matched to at most b hyperedges. We show that, for $b = \Omega(\log k)$, any fractional algorithm can be converted to a randomized integral algorithm while incurring a small loss in the competitive ratio.

Let $\mathcal{A}$ be a fractional algorithm that is ρ-competitive and let $\mathcal{H} = (V, W, H)$ be an online k-uniform hypergraph instance. We denote by $x \in [0,1]^H$ the fractional solution constructed by $\mathcal{A}$ on the instance $\mathcal{H}$. The rounding algorithm is now quite simple and is similar to the methods used in [EOJ12, RT87, SS95].

Fix some small $0 < \epsilon < 1$ and initialize two empty sets of hyperedges $S, \mathcal{M} \leftarrow \emptyset$. Upon the arrival of an online vertex $w \in W$ with $\delta(w) \subseteq H$ and $x_h \in [0,1]$ for every $h \in \delta(w)$, the rounding algorithm is as follows:

- For all $h \in \delta(w)$, independently add h to S with probability $x'_h := (1-\epsilon)x_h$.
- If h was added to S, add it to $\mathcal{M}$ as long as it does not violate the degree constraints.

The solution outputted is $\mathcal{M} \subseteq H$. Let us denote this rounding algorithm by $R(\mathcal{A}, \epsilon)$.

Lemma 4.8.1. *Let $\mathcal{A}$ be a ρ-competitive fractional algorithm. The randomized integral algorithm $R(\mathcal{A}, \epsilon)$ achieves a competitive ratio of at least $(1-\epsilon)(1 - k\exp(-\epsilon^2 b/3)) \cdot \rho$.*

Proof:
Consider an arbitrary node $v \in V \cup W$. To bound the probability that v is matched to more than b hyperedges in S, we use a Chernoff bound [Doe19, Theorem 1.10.1]. Fix a node v and a hyperedge h, and let $X_{v,h} = \sum_{h' \in \delta(v) \setminus \{h\}} \mathbf{1}_{\{h' \in S\}}$. Note that

$\mu := \mathbb{E}[X_{v,h}] \leq (1-\epsilon)b$. If $\mu = 0$, it is clear that $x_{v,h} \leq b$, so assume $\mu > 0$. We now have:

$$\begin{aligned}\Pr\left[X_{v,h} \geq b\right] &\leq \exp\left(-\min\left(\left(\frac{b-\mu}{\mu}\right)^2, \frac{b-\mu}{\mu}\right)\mu/3\right) \\ &\leq \exp\left(-\min\left(\frac{(b-\mu)^2}{\mu}, b-\mu\right)/3\right) \\ &\leq \exp\left(-\min\left(\epsilon^2 b, \epsilon b\right)/3\right) \leq \exp(-\epsilon^2 b/3),\end{aligned}$$

where the second inequality follows from $b - \mu \geq \epsilon b$ and the last inequality from $b/\mu \geq 1$. We now upper bound the probability that a hyperedge h cannot be included in $\mathcal{M}$ because of the degree constraints:

$$\Pr[h \in S \setminus \mathcal{M} \mid h \in S] \leq \sum_{v \in h} \Pr\left[X_{v,h} \geq b\right] \leq k \exp(-\epsilon^2 b/3).$$

Hence, we have:

$$\begin{aligned}\mathbb{E}[|\mathcal{M}|] = \sum_{h \in H} \Pr[h \in \mathcal{M}] &= \sum_{h \in H} \Pr[h \in S]\left(1 - \Pr[h \in S \setminus \mathcal{M} \mid h \in S]\right) \\ &\geq \sum_{h \in H} x'_h \left(1 - k\exp(-\epsilon^2 b/3)\right) \\ &= \left(1 - k\exp(-\epsilon^2 b/3)\right)(1-\epsilon)\sum_{h \in H} x_h \\ &\geq \left(1 - k\exp(-\epsilon^2 b/3)\right)(1-\epsilon)\rho\, \mathsf{OPT}_{\mathsf{LP}}.\end{aligned}$$

□

If $b = C \cdot \log(k)$ for some $C > 6$, then by choosing $\epsilon = \sqrt{6/C}$ we get that the competitive ratio is at least $(1 - \sqrt{6/C})(1 - \frac{1}{k})\rho$. By using the $\Omega(1/\log k)$-competitive fractional algorithm from [BN09], this gives an $\Omega(1/\log k)$-competitive integral algorithm for this setting.

Chapter 5

Price of anarchy for scheduling games via vector fitting

In this chapter, we introduce a semidefinite programming relaxation allowing to relax scheduling and congestion problems which can be cast as a binary integer quadratic program, an example of which we have introduced in (2.1.2). We then provide a dual fitting technique on this SDP allowing to tightly upper bound the price of anarchy of these games in a simple and unified way. We also illustrate how the same technique can be adapted to tightly upper bound the approximation ratio of local search algorithms.

In the next chapter, we will show how this technique can be adapted to analyze the competitive ratio of different online algorithms for such scheduling problems.

5.1 Introduction

A standard way of quantifying inefficiency of selfish behaviour in algorithmic game theory is the price of anarchy, introduced in [KP99]. It is defined as the ratio between the cost of a worst-case Nash equilibrium and the cost of a social optimum. This definition can be used to understand inefficiency of pure or mixed Nash equilibria, and can also be extended to more general notions, such as correlated or coarse-correlated equilibria.

Developing tools to bound the price of anarchy is a central question, and several approaches have been proposed in the literature to tackle this problem. One technique that has been very successful for a variety of games is the smoothness framework, introduced in [Rou15]. One advantage of this approach is that it automatically bounds the price of anarchy for all the different notions of equilibria mentioned above, yielding bounds on the robust price of anarchy of a game [Rou15].

Another possible avenue is to use convex relaxations to help bound the price of anarchy, as done in [KM14]. The high-level approach is to formulate a convex relaxation of the underlying optimization problem of a given game, and to con-

struct a feasible solution to the dual of that relaxation, whose cost can then be compared to the cost of an equilibrium. Bounding the ratio between the cost of the equilibrium and of the feasible dual solution then yields an upper bound on the price of anarchy by weak duality.

In this chapter, we build on this approach and show that a single convex semidefinite programming relaxation can be used to obtain tight (robust) price of anarchy bounds for several different congestion and scheduling games. This relaxation can in fact be obtained using the first round of the Lasserre hierarchy [Las01], and the proofs bounding the price of anarchy through the dual of that relaxation are surprisingly simple and essentially follow the same template for all the games considered. In addition to bounding the price of anarchy, it turns out that the same approach also allows to bound the approximation ratio of local optima for machine scheduling.

As a main illustration of this technique, we consider the following model of congestion games. We are given a set of players N and a set of resources E. The strategy set for each player $j \in N$ is a collection of subsets of resources and is denoted by $\mathcal{S}_j \subseteq 2^E$. Each player has a resource-dependent processing time $p_{ej} \geq 0$ and a weight $w_j \geq 0$. Once each player chooses a strategy, if a given resource $e \in E$ is shared by several players, then e uses a *coordination mechanism*, defined as a local policy for each resource, in order to process the players using it. One natural example of such a coordination mechanism is to order the players by increasing Smith ratios, defined as the ratio between the processing time on a resource and the weight of a given player [Smi56].

This model is a generalization of the unrelated machine scheduling game $R||\sum w_j C_j$, where each job needs to selfishly pick a machine to minimize its own weighted completion time, while knowing that each machine uses a coordination mechanism to process the jobs assigned to it. In our model, the set of resources E is the set of machines, and the strategy set of each player is a subset of the machines. An important special case of our model, which generalizes $R||\sum w_j C_j$, is the following selfish routing game. We are given a directed graph $G = (V, E)$ and a set of players N. Each player j wants to pick a path between a source node $s_j \in V$ and a sink node $t_j \in V$. The strategy set $\mathcal{S}_j$ for player $j \in N$ is the set of all paths between s_j and t_j. A parallel link network where each player has the same source and sink node exactly corresponds to the $R||\sum w_j C_j$ scheduling problem.

The work of [CCG+11] considers three different coordination mechanisms for $R||\sum w_j C_j$. Their main results are that *Smith's Rule* leads to a tight price of anarchy of 4, and this can be improved to $(3+\sqrt{5})/2 \approx 2.618$ and $32/15 \approx 2.133$ by respectively considering a preemptive mechanism called *Proportional Sharing*, as well as a randomized one named *Rand*. The latter two results in fact bound the *coordination ratio* of the coordination mechanism, meaning that the cost of a worst-case Nash equilibrium is compared to the cost of an optimal solution under *Smith's Rule*, since this is always how an optimal solution processes the jobs once

an assignment is given [Smi56]. The proof technique they use to obtain their results is based on the smoothness framework [Rou15]. In order to exploit the structure of the problem, they map strategy vectors into a carefully chosen inner product space, where the social cost is closely related to a squared norm in that space. Generalizing their results to selfish routing games was mentioned as an open question.

The inner product space structure developed in [CCG+11] turns out to have a natural connection to semidefinite programming, since the latter can be seen as optimizing over inner products of vectors. In this work, we study this connection and show that it leads to simple dual fitting proofs that allow to tightly bound the price of anarchy, as well as the approximation ratio of local optima, for several different congestion and scheduling games in a unified way. We hope that this new approach might turn out to be useful in other contexts as well.

Our contributions

Our main contribution is a unified dual fitting technique on a single semidefinite program to bound the price of anarchy, as well as the approximation ratio of local optima, for a class of games whose underlying optimization problem can be cast as a binary quadratic program. We illustrate the applicability of this approach for different scheduling and congestion games. The semidefinite program used can be obtained by applying one round of the Lasserre/Sum of Squares hierarchy to the exact binary quadratic program.

We show that the three bounds of respectively $4, (3+\sqrt{5})/2 \approx 2.618$ and $32/15 \approx 2.133$ for the policies *Smith's Rule*, *Proportional Sharing* and *Rand* can be obtained through our approach in the above congestion game model. This yields alternative and simple proofs of these results in a more general model, which avoid the use of minimum norm distortion inequalities, as done in [CCG+11]. We moreover show that the last bound can be improved from 2.133 to 2 for the natural special case where the processing times are proportional to the weight of a given player on every feasible resource. This means that every resource has a real-value $\lambda_e \geq 0$, interpreted as the processing power, and the processing time of every player satisfies $p_{ej} \in \{\lambda_e w_j, \infty\}$ for every $e \in E, j \in N$. The importance of this model in a scheduling setting has been mentioned in [KST17]. This improvement from 2.133 to 2 can also be obtained for general instances if one considers the price of anarchy of the game, rather than the coordination ratio. This means that the cost of a worst-case Nash equilibrium is now compared against an optimal solution using the *Rand* policy, rather than *Smith's Rule*.

Moreover, we show that the same approach (on the same relaxation) can be used to bound the approximation ratio of local optima of local search algorithms for machine scheduling under the sum of weighted completion times objective. We first consider a natural algorithm whose local optima simply ensure that no job can decrease the global objective function by switching to a different

machine. Observe the analogy with Nash equilibria, which ensure that no job can improve its own objective (or completion time) by switching machines. We recover the approximation ratios of $(3+\sqrt{5})/2 \approx 2.618$ and $(5+\sqrt{5})/4 \approx 1.809$ for the scheduling problems $R||\sum w_j C_j$ and $P|\mathcal{M}_j|\sum w_j C_j$ given in [CM22]. In addition, we also analyze an improved local search algorithm for $R||\sum w_j C_j$ attaining a bound of $(5+\sqrt{5})/4+\varepsilon \approx 1.809+\varepsilon$ [CGV17], and show an almost matching lower bound of 1.791. To the best of our knowledge, this is the currently best known *combinatorial* approximation algorithm for this problem.

As a further illustration of the technique, we apply it to two classical games and show that it yields simple proofs of known tight bounds. We first show how to get the tight bound of $(3+\sqrt{5})/2$ for the price of anarchy of weighted affine congestion games. While a dual fitting proof through a convex relaxation of this bound is already provided in [KM14], this result showcases the versatility of our SDP relaxation and of the fitting strategy. In addition, a dual fitting proof of the Kawaguchi-Kyan bound of $(1+\sqrt{2})/2$ for the pure price of anarchy of the scheduling game $P||\sum w_j C_j$ is also provided through the same relaxation. We note that the dual fitting strategy used for this result uses a reduction to worst-case instances of [Sch11].

Further related work

There is a vast literature on exact or approximation algorithms for scheduling problems under the (weighted) sum of completion times objective. We adopt the standard three-field notation $\alpha|\beta|\gamma$ of [GLLK79]. The problem with unweighted completion times $R||\sum C_j$ is polynomial time solvable [Hor73, BCJS74]. For $P||\sum C_j$ on parallel machines, the shortest first policy gives an optimal solution which also turns out to be a Nash equilibrium [CM67]. On the other hand, the weighted completion times objective is NP-hard even for $P||\sum w_j C_j$ [LKB77]. A PTAS is known for $P||\sum w_j C_j$ [SW00], while $R||\sum w_j C_j$ is APX-hard [HSW98]. Constant factor approximation algorithms are however possible, with major results being a simple 3/2-approximation by rounding a convex relaxation [Sku01, SS99] and the first algorithm breaking the 3/2-approximation using a semidefinite relaxation [BSS16]. We note that the primal semidefinite program that we use is very similar to their relaxation. Building on this, subsequent improvements have been made [IS20, IL23, Har24] with the current best (to the best of our knowledge) approximation algorithm for this problem obtaining a ratio of $1.36+\varepsilon$ [Li24]. In the special case where Smith ratios are uniform, an improved bound of $(1+\sqrt{2})/2+\varepsilon$ has been obtained [KST17].

Scheduling problems have also been vastly studied from a game theoretic perspective. For $P||\sum w_j C_j$, the pure price of anarchy of *Smith's Rule* coincides with the approximation ratio of a simple greedy algorithm and was shown to be $(1+\sqrt{2})/2 \approx 1.207$ in a classic result of [KK86]. A much simpler proof of this result is shown in [Sch11]. Interestingly, the mixed price of anarchy of this game

is higher, with a tight bound of 3/2 even for $P||\sum C_j$ [RS13]. For the unweighted version, *Smith's Rule* in fact reduces to the shortest processing time first policy, under which [HU11] shows an upper and lower bound of respectively 2 for the robust price of anarchy and $e/(e-1) \approx 1.58$ for the pure price of anarchy of $Q||\sum C_j$. For related machines, it is still an interesting open question whether the upper bounds of respectively 2 and 4 for $Q||\sum C_j$ and $Q||\sum w_j C_j$ can be improved.

Coordination mechanisms were introduced in the work of [CKN04] for $P||C_{max}$ and a selfish routing/congestion game. Four different scheduling games under four different policies were analyzed in [ILMS09] under the makespan objective. Upper and lower bounds for different coordination mechanisms for $R||C_{max}$ can be found in [AJM08, Car13, FS10, CDNK11, AH12]. Further work on coordination mechanisms for the makespan objective has been done in [BIKM14, CF19, Kol13].

The literature for the sum of completion times objective is somewhat sparser. The work of [CCG+11] considers $R||\sum w_j C_j$ and shows that the policies *Smith's Rule*, *Proportional Sharing* and *Rand* respectively give bounds of $4, 2.618$ and 2.133 on the robust price of anarchy. The first two bounds are tight, with matching lower bounds given in [CQ12] and [CFK+06]. The latter two coordination mechanisms can in fact be interpreted as a cost-sharing protocol [CGV17]. Using similar techniques, [ACH14] extend some of the previous results to multi-job scheduling games. Coordination mechanisms for a more general model with release dates and assignment costs have been studied in [BIKM14].

The study of the price of anarchy for weighted congestion games was initiated in [KP99] for parallel links under the maximum load (or makespan) objective. Tight bounds for parallel links have been shown in [CV07]. For general networks under the MinSum objective with affine latency functions, the works of [AAE05, CK05] establish that the price of anarchy is 5/2 for the unweighted version and $(3+\sqrt{5})/2$ in the weighted case. Other models have been studied in [ADG+11, CFK+06, BGR14, STZ04, FOV08]. To the best of our knowledge, the literature on coordination mechanisms for congestion/selfish routing games is relatively sparse [CKN04, CMP14, BKM14].

5.2 Preliminaries

Game format. All the games considered are of the following form. A set of players N is given. Each player $j \in N$ has a strategy set $\mathcal{S}_j$, and we denote by $x_{ij} \in \{0,1\}$ the binary value indicating whether the player chooses strategy $i \in \mathcal{S}_j$. If $x_{ij} \in [0,1]$, then this corresponds to the probability of player j independently choosing strategy i. The (expected) cost incurred by the player is denoted by $C_j(x)$ and is a *quadratic* (possibly non-convex) function of x. Given weights $w_j \geq 0$ for every player $j \in N$, the total social cost is the weighted sum of costs incurred by every player, and we denote it by $C(x) = \sum_j w_j C_j(x)$.

Scheduling. One example falling in this class are scheduling games. Given is a set of jobs $J = N$, which are the players, and a set of machines M. The strategy set of every player is a subset of the machines $\mathcal{S}_j \subseteq M$. We adopt the standard three-field notation $\alpha|\beta|\gamma$ of [GLLK79], with α denoting the machine environment, β denoting the constraints, and γ denoting the objective function. The most general such problem we consider is $R||\sum w_j C_j$, where each job $j \in N$ has unrelated processing times $p_{ij} \in \mathbb{R}_+ \cup \{\infty\}$ for each $i \in M$. If $p_{ij} = \infty$, we will without loss of generality assume that $i \notin \mathcal{S}_j$. Once an assignment x is fixed, the optimal way to process the jobs for each machine is to order them by increasing Smith ratios, which we denote as $\delta_{ij} := p_{ij}/w_j$. We denote $k \prec_i j$ if k precedes j in the ordering of machine i, meaning that $\delta_{ik} \leq \delta_{ij}$. We assume ties are broken in a consistent way. The completion time of every job is then

$$C_j(x) = \sum_{i \in M} x_{ij} \Big(p_{ij} + \sum_{k \prec_i j} p_{ik} x_{ik} \Big).$$

Observe that this is indeed a quadratic function in x. If every job has the same processing time $p_{ij} = p_j$ on every machine, this model is denoted by $P||\sum w_j C_j$. If $p_{ij} \in \{p_j, \infty\}$, then the model is denoted as $P|\mathcal{M}_j|\sum w_j C_j$.

Congestion model. We consider the following model of congestion games, which generalize the scheduling games described above. We are given a set of players N and a set of resources E. The strategy set for each player $j \in N$ is denoted by $\mathcal{S}_j \subseteq 2^E$ and is a collection of subsets of resources. Each player has a resource-dependent processing time $p_{ej} \geq 0$ and a weight $w_j \geq 0$. Without loss of generality, we assume that for every feasible strategy $i \in \mathcal{S}_j$ of a player $j \in N$, we have that $p_{ej} < \infty$ for every $e \in i$ (otherwise simply remove i from $\mathcal{S}_j$ since it is not a valid strategy). The Smith ratio is defined as $\delta_{ej} = p_{ej}/w_j$ for every $e \in E, j \in N$. We denote $k \prec_e j$ if $\delta_{ek} \leq \delta_{ej}$, meaning that k has a smaller Smith ratio than j on the resource $e \in E$, where ties are broken in a consistent manner. For a given assignment $(x_{ij})_{j \in N, i \in \mathcal{S}_j}$, we denote

$$z_{ej} := \sum_{i \in \mathcal{S}_j : e \in i} x_{ij}.$$

We invite the reader to think about pure assignments. In that case, $x_{ij} \in \{0,1\}$ is binary and indicates whether or not player j chooses strategy $i \in \mathcal{S}_j$, whereas $z_{ej} \in \{0,1\}$ takes value one if j uses the resource $e \in E$, i.e. chooses a strategy $i \in \mathcal{S}_j$ containing resource $e \in E$. In the case of mixed assignments, $x_{ij} \in [0,1]$ represents the probability of player j independently choosing strategy i, whereas $z_{ej} \in [0,1]$ represents the probability of player j using resource e. Once an assignment x is fixed, Smith's Rule is again the optimal way for every resource to process the jobs, and the cost incurred by a player $j \in N$ is given by:

$$C_j(x) = \sum_{i \in S_j} x_{ij} \sum_{e \in i} \Big(p_{ej} + \sum_{k \prec_e j} p_{ek}\, z_{ek} \Big).$$

Nash equilibria. An assignment x is a Nash equilibrium if no player can get a lower cost by changing his/her strategy. The price of anarchy is defined as the ratio between the cost of a worst-case Nash equilibrium and the cost of an optimal solution. Unless explicitly stated otherwise, we consider mixed Nash equilibria, meaning that the following set of constraints is satisfied:

$$C_j(x) \leq C_j(x_{-j}, i) \qquad \forall j \in N, \forall i \in \mathcal{S}_j \tag{5.2.1}$$

where x_{-j} refers to the assignment of all players other than j. In Section 5.8, we show how to extend our results to a more general equilibrium concept, namely a *coarse-correlated* equilibrium.

Coordination mechanisms. In the scheduling setting, a coordination mechanism is a set of local policies, one for each machine, deciding on how the jobs assigned to it should be processed. Smith's Rule is one example of such a policy, which is in fact optimal in terms of the social cost. However, picking a different policy may help improve the price of anarchy. One policy that we consider is a preemptive mechanism called *Proportional Sharing*, where the jobs are scheduled in parallel, with each uncompleted job receiving a fraction of the processor time proportional to its weight. Another mechanism is *Rand*, which orders the jobs randomly by ensuring that the probability of job j being scheduled before k is $\delta_{ik}/(\delta_{ij} + \delta_{ik})$ for every pair of jobs j, k. The reader is referred to [CCG+11] for details. In our congestion model, each resource uses one of these coordination mechanisms to process the players using that resource. Note that this modifies the cost $C_j(x)$ incurred by every player, and thus also the social cost $C(x)$.

Coordination ratio and price of anarchy. We make a distinction between the coordination ratio of a coordination mechanism and the price of anarchy of the game. The coordination ratio measures the ratio between a worst-case Nash equilibrium and an optimal solution if every resource uses Smith's Rule to process the players. In contrast, the price of anarchy of the game compares to a weaker optimal solution where each resource uses the chosen mechanism to process the players.

Outline of the chapter. The semidefinite programming relaxation and a high-level view of the approach is presented in Section 5.3. The analysis of the coordination ratio and the price of anarchy of *Smith's Rule*, *Proportional Sharing* and *Rand* for our congestion model are presented in Section 5.4. The analysis of local optima for machine scheduling is done in Section 5.5. The analysis for the price of anarchy of weighted affine congestion games is shown in Section 5.6. The pure price of anarchy of $P||\sum w_j C_j$ is presented in Section 5.7.

5.3 The semidefinite programming relaxation

5.3.1 The primal-dual pair

We assume in this section some basics on semidefinite programs (SDPs), which can be found in Section 2.4. Let N be a set of players, with each player $j \in N$ having a strategy set $\mathcal{S}_j$. An exact quadratic program to compute the social optimum is given by

$$\begin{aligned} \min\, & C(x) \\ & \sum_{i \in \mathcal{S}_j} x_{ij} = 1 \qquad \forall j \in N \\ & x_{ij} \in \{0,1\} \qquad \forall j \in N, \forall i \in \mathcal{S}_j. \end{aligned}$$

Since we assume $C(x)$ to be quadratic in x, the social cost can be written as

$$C(x) = C_{\{0,0\}} + 2 \sum_{j \in N, i \in \mathcal{S}_j} C_{\{0,ij\}}\, x_{ij} + \sum_{\substack{j,k \in N \\ i \in \mathcal{S}_j, i' \in \mathcal{S}_k}} C_{\{ij,\, i'k\}} x_{ij}\, x_{i'k} \tag{5.3.1}$$

for some symmetric matrix C of dimension $1+\sum_{j \in N} |\mathcal{S}_j|$, which has one row/column corresponding to each x_{ij}, in addition to one extra row/column that we index by 0. The above equation (5.3.1) can be written in a compact way as $C(x) = \langle C, X \rangle := \mathrm{Tr}(C^T X)$, where X is the rank one matrix $X = (1,x)(1,x)^T$, where the notation $(1,x)$ refers to a vector in dimension $1 + \sum_{j \in N} |\mathcal{S}_j|$ obtained by appending a coordinate with value 1 to x.

We now consider a semidefinite convex relaxation of the above quadratic program, which can essentially be obtained through the Lasserre/Sum of Squares hierarchy [Las01]. The variable of the program is a positive semidefinite matrix X of dimension $1 + \sum_{j \in N} |\mathcal{S}_j|$, which has one row/column corresponding to each x_{ij}, in addition to one extra row/column that we index by 0.

$$\begin{aligned} \min \langle C, X \rangle & \\ \sum_{i \in \mathcal{S}_j} X_{\{ij,\, ij\}} &= 1 && \forall j \in N \\ X_{\{0,0\}} &= 1 && \\ X_{\{0,\, ij\}} &= X_{\{ij,\, ij\}} && \forall j \in N, i \in \mathcal{S}_j \\ X_{\{ij,\, i'k\}} &\geq 0 && \forall (i,j), (i',k) \text{ with } j,k \in N \\ X &\succeq 0 && \end{aligned}$$

To see that this is in fact a relaxation to the previous quadratic program, note that for any binary feasible assignment x, the rank-one matrix $X = (1,x)(1,x)^T$ is a feasible solution to the SDP with the same objective value. The key observation

that makes this work is the fact that $x_{ij}^2 = x_{ij}$ for $x_{ij} \in \{0,1\}$, leading to $X_{\{ij,\,ij\}} = x_{ij}^2 = x_{ij} = X_{\{0,\,ij\}}$. The dual to this relaxation, written in vector form, is the following. The computation of the dual is shown in Section 5.9.1. We call this relaxation *(SDP-C)*.

$$\begin{aligned} \max \sum_{j \in N} y_j - \frac{1}{2}\|v_0\|^2 & & (5.3.2)\\ y_j \leq C_{\{ij,\,ij\}} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle & \qquad \forall j \in N, i \in \mathcal{S}_j \\ \langle v_{ij}, v_{i'k}\rangle \leq 2\, C_{\{ij,\,i'k\}} & \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in N \end{aligned}$$

The variables of this program are real-valued $y_j \in \mathbb{R}$ for every $j \in N$, as well as vectors $v_0 \in \mathbb{R}^d$ and $v_{ij} \in \mathbb{R}^d$ for every $j \in N, i \in \mathcal{S}_j$ in some dimension $d \in \mathbb{N}$. This will be the relaxation used for every dual fitting argument in this thesis. Depending on the problem we are considering, the matrix C, which only depends on the total social cost, is then picked accordingly. The computation of this matrix for every game considered is shown in Section 5.9.2.

5.3.2 High-level view of the approach and intuition of the dual

We describe here a high-level view of the dual fitting approach and of its main ideas. We also provide some intuition in how the dual program (5.3.2) is used. For clarity, we illustrate the concepts on a simple concrete toy example that we introduced in Section 2.6.3: a weighted load balancing game, which is a special case of an affine weighted congestion game later analyzed in full detail in Section 5.6. Let us remind the reader of the setting.

Example: load balancing. We are given a set of players N and a set of resources E. The strategy set of every player $j \in N$ is a subset of resources $\mathcal{S}_j \subseteq E$ with unrelated weights $w_{ij} \geq 0$ associated for every $i \in \mathcal{S}_j$. Consider a pure assignment x, the *load* of a resource $i \in E$ is defined as the total weight of players assigned to it and is formally defined as $\ell_i(x) = \sum_{j \in N} w_{ij} x_{ij}$. The cost of a player j is then defined as $C_j(x) = \sum_{i \in E} \ell_i(x)\, w_{ij}\, x_{ij}$, meaning that it is the weight multiplied by the load of the resource picked. The social cost can be written as follows

$$C(x) = \sum_{j \in N} C_j(x) = \sum_{i \in E} \sum_{j,k \in N} w_{ij} w_{ik} x_{ij} x_{ik} = \sum_{\substack{j,k \in N \\ i \in \mathcal{S}_j, i' \in \mathcal{S}_k}} w_{ij} w_{ik} x_{ij} x_{i'k} \mathbb{1}_{\{i=i'\}}. \quad (5.3.3)$$

Note that the social cost can also be written in a simple way as $C(x) = \sum_{i \in E} \ell_i(x)^2$. The above equation is however written in the form (5.3.1).

Specializing the dual SDP. After understanding what the social cost looks like as a quadratic function in the form (5.3.1), we are able to write down the dual program (5.3.2) for a considered game. In our example, the above equation tells us that the matrix C has diagonal entries $C_{\{ij,\, ij\}} = w_{ij}^2$ and non-diagonal entries $C_{\{ij,\, i'k\}} = w_{ij}\, w_{ik}\, \mathbb{1}_{\{i=i'\}}$, meaning that we can write down the dual as:

$$\max \sum_{j \in N} y_j - \frac{1}{2}\|v_0\|^2 \tag{5.3.4}$$

$$y_j \leq w_{ij}^2 - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle \qquad \forall j \in N, \forall i \in \mathcal{S}_j \tag{5.3.5}$$

$$\langle v_{ij}, v_{i'k}\rangle \leq 2\, w_{ij}\, w_{ik}\, \mathbb{1}_{\{i=i'\}} \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in N. \tag{5.3.6}$$

Given any Nash equilibrium (or local optimum) x, the goal is to use this dual program to construct a feasible solution with objective value at least $\rho\, C(x)$ for some $\rho \in [0,1]$. By weak duality, this would directly imply an upper bound of $1/\rho$ for the price of anarchy (or approximation ratio).

Correspondence between the SDP constraints and the Nash conditions. The key insight is that the first set of constraints (5.3.5) of the SDP has the same structure as that of the Nash equilibria inequalities (5.2.1). Our goal is to pick a fitting which will ensure that this set of constraints corresponds to (or is implied by) these equilibrium conditions. Fix a Nash equilibrium x and let us write down what the Nash conditions imply for our toy example:

$$C_j(x) \leq w_{ij}\, (\ell_i(x) + w_{ij}) = w_{ij}^2 + w_{ij}\, \ell_i(x) \qquad \forall j \in N, \forall i \in \mathcal{S}_j.$$

A natural way to achieve the desired correspondence is to have the following:

$$y_j \sim C_j(x) \quad , \quad w_{ij}^2 - \frac{1}{2}\|v_{ij}\|^2 \sim w_{ij}^2 \quad , \quad -\langle v_0, v_{ij}\rangle \sim w_{ij}\, \ell_i(x) \tag{5.3.7}$$

where the $\sim$ notation indicates that both quantities are within a fixed constant (which should be the same for all three cases above) of each other. For local search algorithms, the Nash inequalities get replaced by local optima conditions.

Picking the vector fitting. Observe that the second correspondence above implies that $\|v_{ij}\|^2 \sim w_{ij}^2$. The second set of SDP constraints (5.3.6) tells us that for $i \neq i'$, one should have $\langle v_{ij}, v_{i'k}\rangle \leq 0$. We will in fact ensure tightness of this constraint by fitting such two vectors to be *orthogonal*. A very natural candidate for the fitting of v_{ij} in our example thus becomes the following choice:

$$v_{ij} \in \mathbb{R}^E \quad \text{ defined as } \quad v_{ij}(e) = \alpha\, w_{ij}\, \mathbb{1}_{\{i=e\}}$$

for some constant $\alpha \in [0, \sqrt{2}]$ to be determined. The upper bound on α follows again from the second set of SDP constraints (5.3.6), since we now get $\langle v_{ij}, v_{i'k} \rangle = \alpha^2 \, w_{ij} \, w_{i'k} \, \mathbb{1}_{\{i=i'\}}$ under our fitting.

How should v_0 now be picked? There are two desirable properties to be satisfied: we want $-\langle v_0, v_{ij} \rangle \sim w_{ij} \ell_i(x)$ as mentioned above, in addition to relating $\|v_0\|^2$ to the social cost $C(x)$, since it appears in the objective function of the SDP. A very natural candidate becomes the following:

$$v_0 \in \mathbb{R}^E \quad \text{defined as} \quad v_0(e) = -\beta \, \ell_e(x)$$

where $\beta \geq 0$ is to be determined. Note that we now indeed get $-\langle v_0, v_{ij} \rangle = \alpha\beta \, w_{ij} \, \ell_i(x)$ and $\|v_0\|^2 = \beta^2 \, C(x)$, since (5.3.3) can be rewritten as $C(x) = \sum_{i \in E} \ell_i(x)^2$.

Optimizing the constants. How should α and β be picked? We have seen that $\alpha \in [0, \sqrt{2}]$ and $\beta \geq 0$. Observe that under our fitting, constraints (5.3.5) now become $y_j \leq (1 - \alpha^2/2) \, w_{ij}^2 + \alpha\beta \, w_{ij} \, \ell_i(x)$. Correspondence (5.3.7) then tells us to set $1 - \alpha^2/2 = \alpha\beta$ and $y_j = \alpha\beta C_j(x)$. The objective value (5.3.4) of the SDP then becomes $(\alpha\beta - \beta^2/2) \, C(x)$. Since we want to pick α and β to maximize the dual objective in order to get the best possible bound on the price of anarchy/approximation ratio, we would want to solve the following optimization problem:

$$\max\{\alpha\beta - \beta^2/2 : 1 - \alpha^2/2 = \alpha\beta, \alpha \in [0, \sqrt{2}], \beta \geq 0\}.$$

Solving this optimization problem would give a price of anarchy bound of $(3 + \sqrt{5})/2$, which is tight in this setting by a lower bound construction of [CFK+06]. At the high-level, this is the approach used to derive the results in this chapter. We invite the reader to keep this intuition even for more complex games.

5.3.3 Different inner product spaces

In order to construct a feasible solution to this SDP, one needs to construct vectors v_0 and $\{v_{ij}\}_{j \in N, i \in \mathcal{S}_j}$ living in a Euclidean space $\mathbb{R}^d$ for some $d > 0$, in addition to real values $\{y_j\}_{j \in N}$ such that both sets of constraints of the SDP are satisfied. Note that the inner product is the standard Euclidean one where, for given $f, g \in \mathbb{R}^d$, it is defined as $\langle f, g \rangle := \sum_{i=1}^d f_i g_i$. For some games, it will be more convenient to work in a different inner product space, as done in [CCG+11]. Let us fix a finite set E, where each $e \in E$ induces a finite set of positive real values $0 = \delta_0^{(e)} \leq \delta_1^{(e)} \leq \cdots \leq \delta_n^{(e)}$. We define the following inner product space:

$$\mathcal{F}(E) := \left\{ f : E \times [0, \infty) \to \mathbb{R}_+ : f(e,t) = \sum_{j=1}^{n} \alpha_{ej} \, \mathbb{1}_{\left\{\delta_{j-1}^{(e)} \leq t \leq \delta_j^{(e)}\right\}}; \; \alpha_{ej} \in \mathbb{R} \right\}.$$

In words, each element satisfies the fact that $f(e,\cdot)$ is a step-function with breakpoints at $\delta_1^{(e)} \leq \cdots \leq \delta_n^{(e)}$ for every $e \in E$. A valid inner product for two $f, g \in \mathcal{F}(E)$ is then given by:

$$\langle f, g\rangle := \sum_{e\in E} \int_0^\infty f(e,t)\; g(e,t)\; dt. \tag{5.3.8}$$

Another inner product space we will consider is the following. Let us fix E to be a finite set and $K \in \mathbb{N}$. For any positive-definite matrix $M \in \mathbb{R}^{K\times K}$, we can consider the space $\mathcal{G}(E,M) := \mathbb{R}^{E\times[K]}$ where the inner product for two $f, g \in \mathcal{G}(E,M)$ is given by:

$$\langle f, g\rangle := \sum_{e\in E} f(e,\cdot)^T\, M\, g(e,\cdot). \tag{5.3.9}$$

We now argue that we can work in these spaces without loss of generality.

Lemma 5.3.1. *Any feasible dual fitting to (SDP-C) obtained in the inner product spaces $\mathcal{F}(E)$ and $\mathcal{G}(E,M)$ can be converted into a feasible dual fitting with the same objective value in a standard Euclidean space $\mathbb{R}^d$ for some $d > 0$ endowed with the standard inner product.*

Proof:
For both spaces, we argue that a collection of elements in it can be mapped to a collection of vectors in a standard Euclidean space while preserving all the pairwise inner products (and thus also preserving the norms). This then clearly implies the claim.

We first show the statement for $\mathcal{F}(E)$. Let us denote the difference between two breakpoints as $\Delta_j^{(e)} := \delta_j^{(e)} - \delta_{j-1}^{(e)}$. For each element $f \in \mathcal{F}(E)$, define $f' \in \mathbb{R}^{E\times[n]}$ as $f'(e,j) := f\Big(e, \delta_{j-1}^{(e)} + \Delta_j^{(e)}/2\Big)\sqrt{\Delta_j^{(e)}}$. By computing the integral of a step function, we clearly have that for $f, g \in \mathcal{F}(E)$,

$$\langle f, g\rangle = \sum_{e\in E}\sum_{j=1}^{n} \Delta_j^{(e)} f\Big(e, \delta_{j-1}^{(e)} + \Delta_j^{(e)}/2\Big)\; g\Big(e, \delta_{j-1}^{(e)} + \Delta_j^{(e)}/2\Big) = \langle f', g'\rangle.$$

Note that the last inner product is the standard Euclidean one, thus showing the claim for $\mathcal{F}(E)$.

We now show the claim for $\mathcal{G}(E,M)$. Let us write a rank-one decomposition of the positive definite matrix $M = \sum_{j=1}^K u_j u_j^T$, which can for instance be done through the spectral decomposition. For each $f \in \mathcal{G}(E,M)$, we define a modified $f' \in \mathbb{R}^{E\times[K]}$ as $f'(e,j) := f(e,\cdot)^T u_j$. For $f, g \in \mathcal{G}(E,M)$, we then have:

$$\langle f, g\rangle = \sum_{e\in E}\sum_{j=1}^{K} f(e,\cdot)^T u_j\; u_j^T g(e,\cdot) = \sum_{e\in E}\sum_{j=1}^{K} f'(e,j) g'(e,j) = \langle f', g'\rangle.$$

□

5.4 Congestion games with coordination mechanisms

5.4.1 Smith's Rule

The first coordination mechanism we consider is Smith's Rule. If x is a mixed assignment, each player first independently picks a strategy according to his/her distribution specified by x to get an assignment. Once an assignment is set, each resource orders the players using it by increasing Smith ratios and processes them in that order. The cost incurred by a player j on a resource e that he/she is using is then $p_{ej} + \sum_{k \prec_{ej}} p_{ek}\, z_{ek}$. The total cost incurred by a player is the sum of the costs incurred on all the resources used. More formally, the completion time of player $j \in N$ under Smith's Rule is defined to be:

$$C_j(x) = \sum_{i \in S_j} x_{ij} \sum_{e \in i} \Big(p_{ej} + \sum_{k \prec_e j} p_{ek}\, z_{ek} \Big). \tag{5.4.1}$$

The outer sum only has one term for a binary assignment. For a mixed assignment x, the expression above is the expected cost, by the law of total probability and independence. The social cost is the sum of weighted completion times:

$$C(x) := \sum_{j \in N} w_j\, C_j(x) = \sum_{e \in E} \sum_{j \in N} w_j\, p_{ej}\, z_{ej} + \sum_{e \in E} \sum_{j \in N, k \prec_e j} w_j\, p_{ek}\, z_{ek}\, z_{ej}, \tag{5.4.2}$$

where the second equality follows by using the definition of $C_j(x)$ and changing the order of summation. Moreover, if x is a Nash equilibrium, the following inequalities are satisfied:

$$C_j(x) \le \sum_{e \in i} \Big(p_{ej} + \sum_{k \prec_e j} p_{ek} z_{ek} \Big) \qquad \forall j \in N,\ \forall i \in \mathcal{S}_j. \tag{5.4.3}$$

The dual semidefinite relaxation (5.3.2) then becomes the following, we call it *(SDP-SR)*. The computation of the cost matrix C in this setting is shown in Section 5.9.2.

$$\max \sum_{j \in N} y_j - \frac{1}{2}\|v_0\|^2 \tag{5.4.4}$$

$$y_j \le \sum_{e \in i} w_j\, p_{ej} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij} \rangle \qquad \forall j \in N, \forall i \in \mathcal{S}_j \tag{5.4.5}$$

$$\langle v_{ij}, v_{i'k} \rangle \le \sum_{e \in i \cap i'} w_j\, w_k\, \min\{\delta_{ej}, \delta_{ek}\} \qquad \forall (i,j) \ne (i',k) \text{ with } j,k \in N \tag{5.4.6}$$

We note that, in order to bound the coordination ratio of a coordination mechanism, one needs to construct a feasible dual solution to this relaxation, since it gives a valid lower bound on the optimal solution. Indeed, once an assignment is fixed, the optimal ordering on every resource is to schedule the players according to Smith's Rule [Smi56].

Theorem 5.4.1. *For any Nash equilibrium x of the above congestion game, where each resource uses the Smith's Rule policy, there exists a feasible (SDP-SR) solution with value at least $1/4\, C(x)$. This implies that the price of anarchy and the coordination ratio is at most 4.*

Remark 5.4.2. This bound is tight, with a matching lower bound given in [CQ12] even for scheduling on restricted identical machines with unit processing times.

Proof:
We assume that the SDP vectors live in the inner product space $\mathcal{F}(E)$. By Lemma 5.3.1, this is without loss of generality. Let us fix $\beta = 1/2$, we now state the dual fitting for *(SDP-SR)*:

- $v_0(e,t) := -\beta \sum_{k\in N} w_k\, z_{ek}\, \mathbb{1}_{\{t\le\delta_{ek}\}}$
- $v_{ij}(e,t) :=\ w_j\, \mathbb{1}_{\{e\in i\}}\, \mathbb{1}_{\{t\le\delta_{ej}\}} \qquad \forall j\in N, \forall i\in\mathcal{S}_j$
- $y_j := \beta\, w_j\, C_j(x) \qquad \forall j\in N.$

Let us now compute the different inner products and norms that we need. For a job $j\in N$ and a strategy $i\in\mathcal{S}_j$, we have

$$\|v_{ij}\|^2 = \sum_{e\in i} w_j^2\, \delta_{ej} = \sum_{e\in i} w_j\, p_{ej}.$$

For v_0, we give an upper bound with respect to $C(x)$:

$$\begin{aligned}
\frac{1}{\beta^2}\,\|v_0\|^2 &= \sum_{e\in E}\sum_{j,k\in N} w_j\, w_k\, z_{ej}\, z_{ek} \int_0^\infty \mathbb{1}_{\{t\le\delta_{ej}\}}\mathbb{1}_{\{t\le\delta_{ek}\}}\, dt \\
&= \sum_{e\in E}\sum_{j,k\in N} w_j w_k z_{ej} z_{ek} \min\{\delta_{ej},\delta_{ek}\} \\
&= \sum_{e\in E}\sum_{j\in N} w_j^2\, z_{ej}^2\, \delta_{ej} + 2\sum_{e\in E}\sum_{j\in N,k\prec_{ej}} w_j w_k z_{ej} z_{ek} \delta_{ek} \\
&= \sum_{e\in E}\sum_{j\in J} w_j\, p_{ej}\, z_{ej}^2 + 2\sum_{e\in E}\sum_{j\in N,k\prec_{ej}} w_j\, p_{ek}\, z_{ej}\, z_{ek} \\
&\le 2\, C(x). \qquad (5.4.7)
\end{aligned}$$

The last equality uses the definition of the Smith Ratio $\delta_{ej} = p_{ej}/w_j$, whereas the last inequality follows from the fact that $z_{ej}^2 \leq z_{ej}$ (since $z_{ej} \in [0,1]$) as well as the definition of the social cost (5.4.2). In addition, for any $(i,j) \neq (i',k)$ with $j,k \in N$, we have

$$\begin{aligned}\langle v_{ij}, v_{i'k} \rangle &= \sum_{e \in E} w_j \, w_k \, \mathbb{1}_{\{e \in i\}} \, \mathbb{1}_{\{e \in i'\}} \int_0^\infty \mathbb{1}_{\{t \leq \delta_{ej}\}} \mathbb{1}_{\{t \leq \delta_{ek}\}} \, dt \\ &= \sum_{e \in i \cap i'} w_j \, w_k \, \min\{\delta_{ej}, \delta_{ek}\} \end{aligned} \tag{5.4.8}$$

and observe that this tightly satisfies the second set of SDP constraints (5.4.6). Finally,

$$\langle v_0, v_{ij} \rangle = -\beta \sum_{e \in i} \sum_{k \in N} w_j \, w_k \, z_{ek} \, \min\{\delta_{ej}, \delta_{ek}\}.$$

Let us now check that this is a feasible solution to *(SDP-SR)*. The second set of constraints is satisfied due to (5.4.8). The first set of constraints (5.4.5) under the above fitting becomes:

$$\begin{aligned} y_j &\leq \sum_{e \in i} w_j \, p_{ej} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij} \rangle \\ &\iff \beta \, w_j \, C_j(x) \leq \frac{1}{2} \sum_{e \in i} w_j \, p_{ej} + \beta \sum_{e \in i} \sum_{k \in N} w_j \, w_k \, z_{ek} \, \min\{\delta_{ej}, \delta_{ek}\} \\ &\iff C_j(x) \leq \sum_{e \in i} \Big(p_{ej} + \sum_{k \in N} w_k \, z_{ek} \, \min\{\delta_{ej}, \delta_{ek}\} \Big) \\ &\iff C_j(x) \leq \sum_{e \in i} \Big(p_{ej} + \sum_{k \prec_e j} p_{ek} \, z_{ek} + \sum_{k \succeq_e j} w_k \, z_{ek} \, \delta_{ej} \Big). \end{aligned}$$

We have simplified both sides by $\beta \, w_j = 1/2 \, w_j$ in the third line, which holds by definition of $\beta := 1/2$. We have also used the definition of the Smith ratio $\delta_{ek} = p_{ek}/w_k$ in the last line. This set of constraints is now clearly satisfied by the Nash conditions (5.4.3). The objective function of this SDP can now be lower bounded using (5.4.7):

$$\sum_{j \in N} y_j - \frac{1}{2}\|v_0\|^2 \geq \beta \sum_{j \in N} w_j \, C_j(x) - \beta^2 C(x) = \big(\beta - \beta^2\big) \, C(x) = \frac{1}{4} \, C(x).$$

□

5.4.2 The Proportional Sharing policy

In this section, we consider a preemptive policy for every resource named *Proportional Sharing*. Once an assignment is fixed, each resource splits its processing

capacity among the uncompleted jobs proportionally to their weights. Given an assignment x, the completion time of player $j \in N$ is defined to be:

$$C_j(x) = \sum_{i \in S_j} x_{ij} \sum_{e \in i} \Big(p_{ej} + \sum_{k \prec_e j} p_{ek}\, z_{ek} + \sum_{k \succ_e j} w_k\, z_{ek}\, \delta_{ej} \Big).$$

For this policy, it is in fact more intuitive to understand the definition by looking at the weighted completion time:

$$w_j C_j(x) = \sum_{i \in S_j} x_{ij} \sum_{e \in i} \Big(w_j p_{ej} + \sum_{k \in N \setminus \{j\}} w_j w_k \min\{\delta_{ej}, \delta_{ek}\}\, z_{ek} \Big).$$

The social cost is the sum of weighted completion times:

$$C(x) := \sum_{j \in N} w_j\, C_j(x) = \sum_{e \in E} \sum_{j \in N} w_j\, p_{ej}\, z_{ej} + 2 \sum_{e \in E} \sum_{j \in N, k \prec_e j} w_j\, p_{ek}\, z_{ek}\, z_{ej}. \quad (5.4.9)$$

Observe that there is now a factor 2 in front of the second term if one compares it to the *Smith Rule* policy. Moreover, if x is a Nash equilibrium, the following inequalities are satisfied:

$$C_j(x) \le \sum_{e \in i} \Big(p_{ej} + \sum_{k \prec_e j} p_{ek} z_{ek} + \sum_{k \succ_e j} w_k\, z_{ek}\, \delta_{ej} \Big) \qquad \forall j \in N,\ \forall i \in \mathcal{S}_j. \quad (5.4.10)$$

We first need a small lemma about two parameters that will play a key role in the dual fitting. The first property will ensure feasibility of the solution, whereas the second one will be the constant in front of the objective function.

Lemma 5.4.3. *Let $\alpha, \beta \ge 0$ be defined as $\alpha^2 := 2/\sqrt{5}$ and $\beta := 1/\alpha - \alpha/2$. The following two properties hold:*

- $1 - \alpha^2/2 = \alpha\beta$
- $\alpha\beta - \beta^2/2 = 2/(3+\sqrt{5})$

Proof:
The first property is immediate by definition of β. For the second property, we get

$$\alpha\beta - \frac{\beta^2}{2} = 1 - \frac{\alpha^2}{2} - \frac{1}{2}\left(\frac{1}{\alpha} - \frac{\alpha}{2}\right)^2 = \frac{3}{2} - \frac{5\alpha^2}{8} - \frac{1}{2\alpha^2} = \frac{2}{3+\sqrt{5}}.$$

□

Theorem 5.4.4. *For any Nash equilibrium x of the above congestion game, where each resource uses the Proportional Sharing policy, there exists a feasible (SDP-SR) solution with value at least $2/(3+\sqrt{5})\, C(x)$, implying that the coordination ratio is at most $(3+\sqrt{5})/2$.*

Remark 5.4.5. This bound is tight, with a matching lower bound given in [CFK+06] even for the price of anarchy of the game.

Proof:
The proof is very similar to the one of Theorem 5.4.1, but with the modified constants $\alpha^2 := 2/\sqrt{5}$ and $\beta := 1/\alpha - \alpha/2$ stated in Lemma 5.4.3. We now state the dual fitting.

- $v_0(e,t) := -\beta \sum_{k\in N} w_k\, z_{ek}\, \mathbb{1}_{\{t\le\delta_{ek}\}}$
- $v_{ij}(e,t) := \alpha\, w_j\, \mathbb{1}_{\{e\in i\}}\, \mathbb{1}_{\{t\le\delta_{ej}\}} \qquad \forall j\in N, \forall i\in\mathcal{S}_j$
- $y_j := \alpha\beta\, w_j\, C_j(x) \qquad \forall j\in N.$

Using the same computations as in Theorem 5.4.1, we compute the different inner products and norms that we need.

- $\|v_0\|^2 = \beta^2 \left(\sum_{e\in E}\sum_{j\in N} w_j\, p_{ej}\, z_{ej}^2 + 2\sum_{e\in E}\sum_{j\in N, k\prec_{ej}} w_j\, p_{ek}\, z_{ej}\, z_{ek} \right) \le \beta^2 C(x)$
- $\|v_{ij}\|^2 = \alpha^2 \sum_{e\in i} w_j\, p_{ej}$
- $\langle v_0, v_{ij}\rangle = -\alpha\beta \sum_{e\in i}\sum_{k\in N} w_j\, w_k\, z_{ek}\, \min\{\delta_{ej},\delta_{ek}\}$
- $\langle v_{ij}, v_{i'k}\rangle = \alpha^2 \sum_{e\in i\cap i'} w_j\, w_k\, \min\{\delta_{ej},\delta_{ek}\}$

The main difference with *Smith's Rule* which allows us to get an improved bound is the fact that the upper bound on the squared norm of v_0 is a factor 2 stronger in this case (see (5.4.7)), due to the new definition of the social cost $C(x)$ given in (5.4.9). To see that this solution is feasible, note that the second set of SDP constraints (5.4.6) is satisfied due to the last computation above and the fact that $\alpha^2 \le 1$. The first set of constraints (5.4.5) under the above fitting reads:

$$
\begin{aligned}
y_j &\le \sum_{e\in i} w_j\, p_{ej} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle \\
&\iff \alpha\beta\, w_j\, C_j(x) \le \left(1-\frac{\alpha^2}{2}\right)\sum_{e\in i} w_j\, p_{ej} + \alpha\beta \sum_{e\in i}\sum_{k\in N} w_j\, w_k\, z_{ek}\, \min\{\delta_{ej},\delta_{ek}\} \\
&\iff C_j(x) \le \sum_{e\in i}\left(p_{ej} + \sum_{k\in N} w_k\, z_{ek}\, \min\{\delta_{ej},\delta_{ek}\}\right) \\
&\iff C_j(x) \le \sum_{e\in i}\left(p_{ej} + \sum_{k\prec_{ej}} p_{ek}\, z_{ek} + \sum_{k\succeq_{ej}} w_k\, z_{ek}\, \delta_{ej}\right).
\end{aligned}
$$

The third equivalence follows from the first property of Lemma 5.4.3. We see that this is satisfied due to the Nash conditions (5.4.10). The objective value of the solution can now be lower bounded as follows:

$$
\sum_{j\in N} y_j - \frac{1}{2}\|v_0\|^2 \ge \alpha\beta \sum_{j\in N} w_j C_j(x) - \frac{\beta^2}{2}\, C(x) = \left(\alpha\beta - \frac{\beta^2}{2}\right) C(x) = \frac{2}{3+\sqrt{5}}\, C(x)
$$

where the last equality follows by the second property of Lemma 5.4.3. □

5.4.3 The Rand policy

In this section, we consider a randomized policy named *Rand*. If x is a mixed assignment, each player first independently picks a strategy according to his/her distribution specified by x. We denote by $N(e) \subseteq N$ the (possibly random) subset of players using resource $e \in E$. Each resource then orders the players using it randomly in a way ensuring that for any pair $j, k \in N(e)$, the probability that j comes after k in the ordering is exactly equal to $\delta_{ej}/(\delta_{ej} + \delta_{ek})$. Such a distribution can be achieved by sampling one player $j \in N(e)$ at random with probability $\delta_{ej}/\sum_{k \in N(e)} \delta_{ek}$, putting that player at the end of the ordering, and repeating this process. The expected completion time of every player is thus given by:

$$C_j(x) = \sum_{i \in S_j} x_{ij} \sum_{e \in i} \Big(p_{ej} + \sum_{k \neq j} \frac{\delta_{ej}}{\delta_{ej} + \delta_{ek}} \, p_{ek} \, z_{ek} \Big).$$

The social cost is the sum of weighted completion times:

$$C(x) := \sum_{j \in N} w_j \, C_j(x) = \sum_{e \in E} \sum_{j \in N} w_j \, p_{ej} \, z_{ej} + \sum_{e \in E} \sum_{j \in N, k \neq j} \frac{\delta_{ej}\delta_{ek}}{\delta_{ej} + \delta_{ek}} \, w_j w_k \, z_{ej} z_{ek}. \tag{5.4.11}$$

Moreover, if x is a Nash equilibrium, the following inequalities are satisfied:

$$C_j(x) \leq \sum_{e \in i} \Big(p_{ej} + \sum_{k \neq j} \frac{\delta_{ej}}{\delta_{ej} + \delta_{ek}} \, p_{ek} \, z_{ek} \Big) \qquad \forall j \in N, i \in \mathcal{S}_j. \tag{5.4.12}$$

We now state a small lemma about some constants that will be important for the fitting. The first property will ensure that our dual fitting is feasible, whereas the second property will be the constant in front of the objective value of our SDP solution, thus determining the coordination ratio.

Lemma 5.4.6. *Let $\alpha, \beta \geq 0$ be defined as $\alpha := 1$ and $\beta := 3/4$. The following two properties hold:*

- $1 - \alpha^2/4 = \alpha\beta$
- $\alpha\beta - \beta^2/2 = 15/32$

Proof:
The proof is immediate. □

Theorem 5.4.7. *For any instance of the above congestion game under the Rand policy, and for any Nash equilibrium x, there exists a feasible (SDP-SR) solution with value at least* $15/32\ C(x)$*. This implies that the coordination ratio is at most* $32/15 \approx 2.133$*.*

Proof:
For simplicity of presentation, let us assume without loss of generality that the processing times are scaled such that the Smith ratios $\delta_{ej} = p_{ej}/w_j$ with $p_{ej} < \infty$ are all integral. Moreover, let us take $K \in \mathbb{N}$ large enough such that $\delta_{ej} \leq K$ for all pairs $(e, j) \in E \times N$ such that $p_{ej} < \infty$. Consider the matrix $M \in \mathbb{R}^{K \times K}$ given by

$$M_{r,s} := \frac{r\ s}{r+s} \qquad \forall r, s \in \{1, \ldots, K\}.$$

A key insight shown in [CCG+11] is that this matrix is positive-definite. By Lemma 5.3.1, we can thus assume that the SDP vectors live in the space $\mathcal{G}(E, M)$. Let α, β be defined as in Lemma 5.4.6, we now state the dual fitting:

- $v_0(e, r) := -\beta \sum_{k \in N} w_k\ z_{ek}\ \mathbb{1}_{\{\delta_{ek} = r\}}$
- $v_{ij}(e, r) := \alpha\ w_j\ \mathbb{1}_{\{e \in i\}}\ \mathbb{1}_{\{\delta_{ej} = r\}} \qquad \forall j \in N, i \in \mathcal{S}_j$
- $y_j := \alpha\beta\ w_j\ C_j(x) \qquad \forall j \in N.$

Let us now compute the different inner products and norms that we need. For every $j \in N, i \in \mathcal{S}_j$:

$$\frac{1}{\alpha^2}\ \|v_{ij}\|^2 = \sum_{e \in i} M_{\{\delta_{ej}, \delta_{ej}\}} w_j^2 = \sum_{e \in i} \frac{\delta_{ej}}{2}\ w_j^2 = \frac{1}{2} \sum_{e \in i} w_j\ p_{ej}.$$

For the squared norm of v_0, we give an upper bound with respect to $C(x)$:

$$\begin{aligned} \frac{1}{\beta^2}\ \|v_0\|^2 &= \sum_{e \in E} \sum_{r,s=1}^{K} M_{r,s}\ v_0(e, r)\ v_0(e, s) = \sum_{e \in E} \sum_{j,k \in N} w_j\ w_k\ z_{ej}\ z_{ek}\ M_{\{\delta_{ej}, \delta_{ek}\}} \\ &= \sum_{e \in E} \sum_{j,k \in N} \frac{\delta_{ej}\delta_{ek}}{\delta_{ej} + \delta_{ek}} w_j\ w_k\ z_{ej}\ z_{ek} \leq C(x). \end{aligned} \tag{5.4.13}$$

where the last inequality holds by (5.4.11) and $z_{ej}^2 \leq z_{ej}$. For any pair $(i, j) \neq (i', k)$ with $j, k \geq 1$:

$$\frac{1}{\alpha^2}\ \langle v_{ij}, v_{i'k} \rangle = \sum_{e \in i \cap i'} M_{\{\delta_{ej}, \delta_{ek}\}} w_j\ w_k = \sum_{e \in i \cap i'} w_j\ w_k\ \frac{\delta_{ej}\ \delta_{ek}}{\delta_{ej} + \delta_{ek}}. \tag{5.4.14}$$

Finally, we have:

$$\frac{-1}{\alpha\beta}\,\langle v_0, v_{ij}\rangle = \sum_{e\in i}\sum_{k\in N} w_j\, w_k\, z_{ek}\, M_{\{\delta_{ej},\delta_{ek}\}} = \sum_{e\in i}\sum_{k\in N}\frac{\delta_{ej}\delta_{ek}}{\delta_{ej}+\delta_{ek}} w_j\, w_k\, z_{ek}$$
$$= w_j \sum_{e\in i}\sum_{k\in N}\frac{\delta_{ej}}{\delta_{ej}+\delta_{ek}}\, p_{ek}\, z_{ek}$$

where the last equality follows by plugging in the definition of $\delta_{ek} = p_{ek}/w_k$.

Let us now check that this solution is indeed feasible for *(SDP-SR)*. The second set of constraints (5.4.6) is satisfied due to (5.4.14), the fact that $\alpha = 1$, as well as observing that $rs/(r+s) \le \min\{r,s\}$ for all $r, s \ge 0$. The first set of constraints (5.4.5) under the above fitting gives:

$$y_j \le \sum_{e\in i} w_j\, p_{ej} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle$$
$$\iff \alpha\beta\, w_j\, C_j(x) \le \left(1-\frac{\alpha^2}{4}\right)\sum_{e\in i} w_j\, p_{ej} + \alpha\beta\, w_j \sum_{e\in i}\sum_{k\in N}\frac{\delta_{ej}}{\delta_{ej}+\delta_{ek}}\, p_{ek}\, z_{ek}$$
$$\iff C_j(x) \le \sum_{e\in i}\left(p_{ej} + \sum_{k\in N}\frac{\delta_{ej}}{\delta_{ej}+\delta_{ek}}\, p_{ek}\, z_{ek}\right).$$

We have simplified both sides by $\alpha\beta\, w_j = (1-\alpha^2/4)w_j$ in the last equivalence, which holds by the first property of Lemma 5.4.6. These inequalities are now clearly satisfied by the Nash conditions (5.4.12), implying that our fitted solution is in fact feasible. The objective value of our solution can be lower bounded as:

$$\sum_{j\in N} y_j - \frac{1}{2}\|v_0\|^2 \ge \alpha\beta \sum_{j\in N} w_j C_j(x) - \frac{\beta^2}{2}C(x) = \left(\alpha\beta - \frac{\beta^2}{2}\right)C(x) = \frac{15}{32}\,C(x)$$

where the last equality follows from the second property of Lemma 5.4.6. □

We now show that this bound can be improved if we consider the natural special case where the processing time of each player is proportional to its weight for every resource. This means that every resource has a real-value $\lambda_e \ge 0$, and the processing time of every player satisfies $p_{ej} \in \{\lambda_e w_j, \infty\}$ for every $e \in E, j \in N$. Observe that this means that the Smith ratios are uniform for the jobs assigned to a resource: $\delta_{ej} = p_{ej}/w_j = \lambda_e$. The only difference with respect to the previous proof will be a change of constants α, β.

Lemma 5.4.8. *Let $\alpha, \beta \ge 0$ be defined as $\alpha := 2/\sqrt{3}$ and $\beta := 1/\sqrt{3}$. The following two properties hold:*

- $1 - \alpha^2/4 = \alpha\beta$

- $\alpha\beta - \beta^2/2 = 1/2$

Proof:
The proof is immediate. □

Theorem 5.4.9. *If the Smith ratios are uniform for every resource, for any instance of the above game and any Nash equilibrium x, there exists a feasible (SDP-SR) solution with value at least $1/2C(x)$. This implies that the coordination ratio of the game is at most 2.*

Proof:
Let α, β be as in Lemma 5.4.8. The only part of the proof of Theorem 5.4.7 which breaks down under these new constants is the fact that the second set of constraints (5.4.6) of the SDP might be violated, since we now have $\alpha^2 = 4/3 > 1$. Indeed (5.4.14) states that:

$$\langle v_{ij}, v_{i'k} \rangle = \alpha^2 \sum_{e \in i \cap i'} w_j \, w_k \, \frac{\delta_{ej} \, \delta_{ek}}{\delta_{ej} + \delta_{ek}}.$$

The proof of Theorem 5.4.7 used the easy observation that $rs/(r+s) \leq \min\{r, s\}$ for every $r, s \geq 0$ to argue feasibility of the solution. Observe that this bound is very close to tight when $s \gg r$ (or vice versa). In the case of uniform Smith ratios, we can get an improved bound since $\delta_{ej} = \delta_{ek} = \lambda_e$:

$$\langle v_{ij}, v_{i'k} \rangle = \alpha^2 \sum_{e \in i \cap i'} w_j \, w_k \, \frac{\lambda_e}{2} \leq \sum_{e \in i \cap i'} w_j \, w_k \, \lambda_e = \sum_{e \in i \cap i'} w_j \, w_k \, \min\{\delta_{ej}, \delta_{ek}\}$$

where the inequality follows since $\alpha^2/2 = 4/6 \leq 1$. By the second property of Lemma 5.4.8, the objective value can now be lower bounded as

$$\sum_{j \in N} y_j - \frac{1}{2}\|v_0\|^2 \geq \alpha\beta \sum_{j \in N} w_j C_j(x) - \frac{\beta^2}{2} C(x) = \left(\alpha\beta - \frac{\beta^2}{2}\right) C(x) = \frac{1}{2} \, C(x).$$

□

We now show that this bound of 2 can also be attained for arbitrary instances if we consider the price of anarchy of the game, rather than the coordination ratio, meaning that we now compare against the optimal solution under the *Rand* policy. More precisely, we compare against the best possible assignment x, whose expected cost is measured if every resource uses the *Rand* policy to process the players. Note that this cost is always higher than the cost if every resource were to use *Smith's Rule*. In that case, a relaxation giving a valid lower bound on the social optimum is the following, we call it *(SDP-RAND)*. The computation of the

cost matrix C to plug-in in (5.3.2) in this setting is once again left to Section 5.9.2.

$$\max \sum_{j\in N} y_j - \frac{1}{2}\|v_0\|^2$$
$$y_j \le \sum_{e\in i} w_j\, p_{ej} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle \qquad \forall j \in N, \forall i \in \mathcal{S}_j$$
$$\langle v_{ij}, v_{i'k}\rangle \le 2 \sum_{e\in i\cap i'} w_j\, w_k\, \frac{\delta_{ej}\,\delta_{ek}}{\delta_{ej}+\delta_{ek}} \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in N$$

Theorem 5.4.10. *For any instance of the above game under the Rand policy, and for any Nash equilibrium x, there exists a feasible (SDP-RAND) solution with value at least $1/2\, C(x)$. This implies that the price of anarchy of the game is at most 2.*

Proof:
The proof is identical to the one of Theorem 5.4.7, but the modified constants α, β stated in Lemma 5.4.8. This new choice of constants is not valid for *(SDP-SR)*, due to the fact that $\alpha^2 > 1$. Indeed, equation (5.4.14) means that the second set of constraints (5.4.6) of *(SDP-SR)* might now be violated. However, the second set of constraints of *(SDP-RAND)* is always satisfied, since $\alpha^2 \le 2$. The objective function guarantee follows from the second property of Lemma 5.4.8. □

5.5 Analyzing local search algorithms for scheduling

We now show that this approach can also be useful to bound the approximation ratio of local search algorithms. We focus on the $R||\sum w_j C_j$ scheduling problem, for which the *(SDP-SR)* relaxation (5.4.4) becomes the following:

$$\max \sum_{j\in J} y_j - \frac{1}{2}\|v_0\|^2$$
$$y_j \le w_j p_{ij} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle \qquad \forall j \in J, \forall i \in \mathcal{S}_j$$
$$\langle v_{ij}, v_{i'k}\rangle \le w_j\, w_k\, \min\{\delta_{ij}, \delta_{ik}\}\, \mathbb{1}_{\{i=i'\}} \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in J.$$

Given an assignment $x \in \{0,1\}^{M\times J}$, the completion time of every job $j \in J$ is:

$$C_j(x) = \sum_{i\in M} x_{ij}\Big(p_{ij} + \sum_{k\prec_i j} p_{ik}x_{ik}\Big).$$

Let us also define the following quantity for every $j \in J$:

$$D_j(x) = \sum_{i \in M} \sum_{k \succ_i j} w_k \, p_{ij} \, x_{ij} \, x_{ik} \tag{5.5.1}$$

and let us denote the weighted sum of processing times as:

$$\eta(x) = \sum_{i \in M} \sum_{j \in J} w_j \, p_{ij} \, x_{ij}. \tag{5.5.2}$$

The total cost can then be written in the following ways:

$$C(x) = \sum_{j \in J} w_j C_j(x) = \eta(x) + \sum_{i \in M} \sum_{j \in J} \sum_{k \prec_i j} w_j \, p_{ik} \, x_{ij} \, x_{ik} \tag{5.5.3}$$

$$C(x) = \eta(x) + \sum_{j \in J} D_j(x) = \eta(x) + \sum_{i \in M} \sum_{j \in J} \sum_{k \succ_i j} w_k \, p_{ij} \, x_{ij} \, x_{ik}. \tag{5.5.4}$$

5.5.1 A simple and natural local search algorithm

A natural and simple local search algorithm for this problem is to move a job from one machine to another if that improves the objective function. If such an improvement is not possible, then a local optimum $x \in \{0,1\}^{M \times J}$ has been reached. Such a local optimum is called a *JumpOpt* in [CM22], and it is shown that the local optimality implies the following constraints. We provide a proof for the sake of completeness.

Lemma 5.5.1. *For any local optimum JumpOpt solution x of the scheduling problem $R||\sum w_j C_j$, the following constraints are satisfied:*

$$w_j C_j(x) + D_j(x) \leq w_j \, p_{ij} + \sum_{k \in J \setminus \{j\}} w_j w_k \min\{\delta_{ij}, \delta_{ik}\} \, x_{ik} \qquad \forall j \in J, \forall i \in M.$$

Proof:
Fix a job j assigned to machine $i^* \in M$ in the local optimum x and let us assume that this job switches to machine $i \in M$. The difference of weighted completion times for job j is

$$w_j \Big(p_{ij} + \sum_{k \prec_i j} p_{ik} x_{ik} \Big) - w_j \Big(p_{i^* j} + \sum_{k \prec_{i^*} j} p_{i^* k} x_{i^* k} \Big).$$

Moreover, the only other jobs for which the completion time is modified are the jobs assigned to i^* and i coming after j in the ordering of the respective machine. Due to the switch of j, these jobs assigned to i^* have their completion time decreased, whereas the ones assigned to i have their completion time increased. The total difference in cost for these jobs is then

$$\sum_{k \succ_i j} w_k \, p_{ij} \, x_{ik} - \sum_{k \succ_{i^*} j} w_k \, p_{i^* j} \, x_{i^* k}.$$

Since x is a local optimum for the global objective function, the total difference in cost (i.e. the sum of the two expressions above) should be non-negative. After rearranging the terms, this is equivalent to

$$w_j\Big(p_{i^*j} + \sum_{k \prec_{i^*} j} p_{i^*k} x_{i^*k}\Big) + \sum_{k \succ_{i^*} j} w_k\, p_{i^*j}\, x_{i^*k} \leq w_j\Big(p_{ij} + \sum_{k \prec_i j} p_{ik} x_{ik}\Big) + \sum_{k \succ_i j} w_k\, p_{ij}\, x_{ik}.$$

Observe that this is exactly the statement of the lemma, finishing the proof. □

We now show that we can recover the tight approximation ratio of $(3+\sqrt{5})/2$ given in [CM22] using our dual fitting approach. Observe the analogy with the proof strategy for the price of anarchy in the previous section. The main difference is that the Nash conditions are replaced by the local optimality conditions of Lemma 5.5.1, and the y variables are fitted differently.

Theorem 5.5.2. *For any JumptOpt local optimum x of the scheduling problem $R||\sum w_j C_j$, there exists a feasible (SDP-SR) solution with value at least $2/(3+\sqrt{5})\, C(x)$.*

Proof:
We assume that the SDP vectors belong to the space $\mathcal{F}(M)$, which is without loss generality by Lemma 5.3.1. Let us fix α, β as in Lemma 5.4.3, i.e. $\alpha^2 := 2/\sqrt{5}$ and $\beta := 1/\alpha - \alpha/2$. We now state the dual fitting:

- $v_0(i,t) := -\beta \sum_{k \in J} w_k\, x_{ik}\, \mathbb{1}_{\{t \leq \delta_{ik}\}}$
- $v_{ij}(i',t) := \alpha\, w_j\, \mathbb{1}_{\{t \leq \delta_{ij}\}}\, \mathbb{1}_{\{i=i'\}} \qquad \forall j \in J, \forall i \in \mathcal{S}_j$
- $y_j := \alpha\beta\,\Big(w_j\, C_j(x) + D_j(x)\Big) \qquad \forall j \in J.$

The desired inner products and norms can be computed to be the following, using essentially the same computations as in the proof of Theorem 5.4.1:

$$\frac{1}{\beta^2}\,\|v_0\|^2 = 2C(x) - \eta(x) \qquad -\frac{1}{\alpha\beta}\langle v_0, v_{ij}\rangle = \sum_{k \in J} w_j\, w_k\, \min\{\delta_{ij}, \delta_{ik}\}\, x_{ik}$$

$$\frac{1}{\alpha^2}\,\|v_{ij}\|^2 = w_j\, p_{ij} \qquad \frac{1}{\alpha^2}\,\langle v_{ij}, v_{i'k}\rangle = w_j\, w_k\, \min\{\delta_{ij}, \delta_{ik}\}\, \mathbb{1}_{\{i=i'\}}.$$

The second set of SDP constraints is satisfied due to the last computation above and the fact that $\alpha^2 \leq 1$. The first set of constraints under this fitting gives:

$$y_j \leq w_j p_{ij} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle$$

$$\iff \alpha\beta\,\Big(w_j\, C_j(x) + D_j(x)\Big) \leq \left(1 - \frac{\alpha^2}{2}\right) w_j\, p_{ij} + \alpha\beta \sum_{k \in J} w_j\, w_k\, x_{ik}\, \min\{\delta_{ij}, \delta_{ik}\}.$$

These are satisfied by Lemma 5.4.3, which states that $1 - \alpha^2/2 = \alpha\beta$, as well as the local optimality conditions of Lemma 5.5.1. The objective function can now be lower bounded as:

$$\sum_{j\in J} y_j - \frac{1}{2}\|v_0\|^2 = \alpha\beta\Big(2C(x) - \eta(x)\Big) - \frac{\beta^2}{2}\Big(2C(x) - \eta(x)\Big)$$
$$= \frac{2}{3+\sqrt{5}}\Big(2C(x) - \eta(x)\Big) \geq \frac{2}{3+\sqrt{5}}\, C(x) \qquad (5.5.5)$$

where the first equality follows from (5.5.3) and (5.5.4), the second equality follows from the second property of Lemma 5.4.3 and the inequality follows from $\eta(x) \leq C(x)$. □

We now show as in [CM22] that one can get an improved bound for the restricted identical machines setting, denoted by $P|\mathcal{M}_j|\sum w_jC_j$. The improvement comes from the fact that for a *JumpOpt* solution x and an optimal solution x^*, we have $\eta(x) = \eta(x^*) = \sum_{j\in J} w_jp_j$ in this setting. This means that, instead of bounding $\eta(x) \leq C(x)$ in the last step of (5.5.5), we can now use the stronger upper bound $\eta(x) \leq C(x^*)$.

Theorem 5.5.3. *For any JumptOpt local optimum x of the scheduling problem $P|\mathcal{M}_j|\sum w_jC_j$, there exists a feasible (SDP-SR) solution with value at least $2/(3+\sqrt{5})\,(2C(x) - C(x^*))$. By weak duality, this implies that the approximation ratio of x is at most $(5+\sqrt{5})/4 \approx 1.809$.*

Proof:
By upper bounding $\eta(x) \leq C(x^*)$ in the last step of (5.5.5), we get the first statement of the theorem. By weak duality, and since the dual solution constructed is feasible, we get that

$$\frac{2}{3+\sqrt{5}}\Big(2C(x) - C(x^*)\Big) \leq C(x^*) \iff \frac{C(x)}{C(x^*)} \leq \frac{5+\sqrt{5}}{4}.$$

□

5.5.2 An improved local search algorithm

In this subsection, we show how our approach allows to analyze an improved local search algorithm for $R||\sum w_jC_j$ by [CGV17] achieving an approximation ratio of $(5+\sqrt{5})/4 + \varepsilon \approx 1.809 + \varepsilon$ for every $\varepsilon > 0$. To the best of our knowledge, this is the best currently known combinatorial approximation algorithm for this problem. We ignore here the issue of the running time and simply analyze the quality of a local optimum, referring the reader to [CGV17] for further details.

Let us fix the constant $\gamma := (9+\sqrt{5})/19 \approx 0.591$. For each job $j \in J$ and an assignment x, we keep a potential function

$$f_j(x) = \sum_{i \in M} x_{ij} \left(w_j \, p_{ij} + \gamma \sum_{k \neq j} w_j w_k \min\{\delta_{ij}, \delta_{ik}\} \, x_{ik} \right) \quad \forall j \in J.$$

If a job $j \in J$ can pick a different machine than the one it is currently on and decrease its potential function $f_j(x)$, then this constitutes an improving move for the local search algorithm. If several improving moves exist, the algorithm picks the one giving the largest decrease in $f_j(x)$. For a local optimum x, we get the following constraints:

$$f_j(x) \leq w_j \, p_{ij} + \gamma \sum_{k \neq j} w_j w_k \min\{\delta_{ij}, \delta_{ik}\} \, x_{ik} \qquad \forall j \in J, \forall i \in M. \tag{5.5.6}$$

As usual with this approach, we first need a small lemma about important constants.

Lemma 5.5.4. *Let $\alpha, \beta, \gamma \geq 0$ be defined as $\alpha^2 = (\sqrt{5}+1)/5$, $\beta^2 = (\sqrt{5}-1)/5$ and $\gamma = (9+\sqrt{5})/19$. The following properties hold:*

- $\alpha\beta/\gamma = 1 - \alpha^2/2$
- $\alpha\beta(2\gamma - 1)/\gamma = \beta^2/2$
- $2\alpha\beta - \beta^2 = 4/(5+\sqrt{5})$

Proof:
The proof consists of simple computations and is omitted. These equations can also be checked on a computer. □

Theorem 5.5.5. *For any local optimum x of the above local search algorithm for $R||\sum w_j C_j$, there exists a feasible (SDP-SR) solution with value at least $4/(5+\sqrt{5})\, C(x)$.*

Proof:
We assume that the SDP vectors belong to the space $\mathcal{F}(M)$, which is without loss of generality by Lemma 5.3.1. Let us fix α, β, γ as in Lemma 5.5.4. We now state the dual fitting:

- $v_0(i,t) := -\beta \sum_{k \in J} w_k \, x_{ik} \, \mathbb{1}_{\{t \leq \delta_{ik}\}}$
- $v_{ij}(i',t) := \alpha \, w_j \, \mathbb{1}_{\{t \leq \delta_{ij}\}} \mathbb{1}_{\{i = i'\}}$ $\qquad \forall j \in J, \forall i \in \mathcal{S}_j$
- $y_j := \frac{\alpha\beta}{\gamma} \, f_j(x)$ $\qquad \forall j \in J$

The desired inner products and norms can be computed to be the following, using essentially the same computations as in the proof of Theorem 5.4.1:

$$\frac{1}{\beta^2}\,\|v_0\|^2 = 2C(x) - \eta(x) \qquad -\frac{1}{\alpha\beta}\langle v_0, v_{ij}\rangle = \sum_{k\in J} w_j\, w_k\, \min\{\delta_{ij}, \delta_{ik}\}\, x_{ik}$$

$$\frac{1}{\alpha^2}\,\|v_{ij}\|^2 = w_j\, p_{ij} \qquad \frac{1}{\alpha^2}\,\langle v_{ij}, v_{i'k}\rangle = w_j\, w_k\, \min\{\delta_{ij}, \delta_{ik}\}\, \mathbb{1}_{\{i=i'\}}.$$

The second set of SDP constraints is satisfied due to the last computation above and the fact that $\alpha^2 \le 1$. The first set of constraints under this fitting gives:

$$\begin{aligned} y_j &\le w_j p_{ij} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle \\ &\iff \frac{\alpha\beta}{\gamma}\, f_j(x) \le \left(1 - \frac{\alpha^2}{2}\right) w_j\, p_{ij} + \frac{\alpha\beta}{\gamma}\gamma \sum_{k\in J} w_j\, w_k\, \min\{\delta_{ij}, \delta_{ik}\}\, x_{ik}. \end{aligned}$$

These are satisfied by Lemma 5.5.4, as well as the local optimality conditions (5.5.6). To argue about the objective, it can be checked (through a simple computation that we omit) that:

$$\sum_{j\in J} f_j(x) = 2\gamma\, C(x) - (2\gamma - 1)\, \eta(x).$$

The objective function then becomes:

$$\begin{aligned} \sum_{j\in J} y_j - \frac{1}{2}\|v_0\|^2 &= \frac{\alpha\beta}{\gamma}\Big(2\gamma\, C(x) - (2\gamma-1)\,\eta(x)\Big) - \frac{\beta^2}{2}\Big(2C(x) - \eta(x)\Big) \\ &= \Big(2\alpha\beta - \beta^2\Big)C(x) - \left(\frac{\alpha\beta\,(2\gamma-1)}{\gamma} - \frac{\beta^2}{2}\right)\eta(x) \\ &= \Big(2\alpha\beta - \beta^2\Big)C(x) = \frac{4}{5+\sqrt{5}}\, C(x) \end{aligned}$$

where the two last equalities follow from Lemma 5.5.4. □

We now provide an almost matching lower bound instance, inspired by constructions in [CFK+06, CM22]. We believe that the upper bound of $(5+\sqrt{5})/4 \approx 1.809$ is tight.

Theorem 5.5.6. *There exists an instance of $R||\sum w_j C_j$ with a local optimum to the above local search algorithm with approximation ratio at least* 1.791.

Proof:
Let $\lambda \approx 1.33849$ be the positive solution to the equation $\lambda^2 = 1 + \gamma\lambda$. We consider an instance with jobs $J = [n]$ and machines $M = [n+1]$. The weights of the jobs are defined as $w_1 = \lambda$ and $w_j = 1/\lambda^{j-1}$ for every $j \ge 2$. The feasible machines

are $\mathcal{S}_j = \{j, j+1\}$ for every $j \in J$ with processing times $p_{1,1}, p_{2,1} = \lambda$ for the first job and $p_{j,j} = \lambda^{j-1}, p_{j+1,j} = \lambda^{j+1}$ for every $j \geq 2$.

The feasible solution where each job j gets assigned to machine j has cost $\sum_{j \in J} w_j \; p_{j,j} \; = \; \lambda^2 + (n-1)$, showing that the optimum solution x^* satisfies $C(x^*) \leq n - 1 + \lambda^2$.

We now claim that the solution x where each job j gets assigned to machine $j+1$ is a local optimum. To see this, observe that the first job clearly cannot decrease his potential function $f_1(x)$ since $p_{1,1} = p_{2,1}$ and no other job is assigned to machine 1 or 2. For $j \geq 2$, we have $f_j(x) = w_j \, p_{j+1,j} = \lambda^2$. If job j were to be reassigned to machine j, then

$$f_j(x_{-j}, j) = w_j \, p_{j,j} + \gamma \, w_{j-1} w_j \min\{\delta_{j,j-1}, \delta_{j,j}\} = 1 + \gamma \, \lambda,$$

which shows that x is a local optimum, by definition of λ. The cost of this solution is then $\sum_{j \in N} \; w_j \, p_{j+1,j} = n\lambda^2$. The approximation ratio of this solution now satisfies

$$\frac{C(x)}{C(x^*)} \geq \frac{n\lambda^2}{n - 1 + \lambda^2} \xrightarrow{n \to \infty} \lambda^2 \approx 1.79154.$$

Picking n large enough thus finishes the proof. □

5.6 Weighted affine congestion games

In this section, we consider the classic weighted affine congestion game. The price of anarchy of this game was settled in [AAE05, CK05] with a tight bound of $(3+\sqrt{5})/2$ and this bound can also be obtained through a dual fitting argument on a convex program [KM14]. We show here how to recover this bound in a simple way through our approach. For simplicity of presentation, we assume in this section that the Nash equilibria considered are pure, extensions to more general equilibrium notions can be found in Section 5.8.

The setting is the following. There is a set N of players and a set E of resources. The strategy set for each player $j \in N$ is denoted by $\mathcal{S}_j \subseteq 2^E$ and is a collection of subsets of resources. Let us also assume that we have unrelated weights $w_{ej} \geq 0$ for every $j \in N, e \in E$. Given a strategy profile x, the *load* of a resource is given by:

$$\ell_e(x) := \sum_{j \in N} w_{ej} \sum_{i \in \mathcal{S}_j \colon e \in i} x_{ij}.$$

The cost incurred by a player j for a pure assignment x is then given by

$$C_j(x) := \sum_{i \in \mathcal{S}_j} x_{ij} \sum_{e \in i} w_{ej} \; (a_e \, \ell_e(x) + b_e)$$

where $a_e, b_e \in \mathbb{R}_{\geq 0}$ for every $e \in E$. The social cost then becomes:

$$C(x) := \sum_{j \in N} C_j(x) = \sum_{e \in E} a_e \, \ell_e(x)^2 + b_e \, \ell_e(x) \tag{5.6.1}$$

where the last equality holds by changing the order of summation and using the definition of $\ell_e(x)$.

The Nash equilibrium conditions imply the following constraints for every $j \in N, i \in \mathcal{S}_j$:

$$C_j(x) \leq \sum_{e \in i} w_{ej} \left(a_e \, (\ell_e(x) + w_{ej}) + b_e \right) = \sum_{e \in i} w_{ej}(a_e \, w_{ej} + b_e) + \sum_{e \in i} w_{ej} \, a_e \, \ell_e(x). \tag{5.6.2}$$

Indeed, if a player $j \in N$ decides to switch to a strategy $i \in \mathcal{S}_j$, then the load on every edge $e \in i$ can go up by at most w_{ej}. The semidefinite relaxation (5.3.2) in this special case becomes the following, we call it *(SDP-CG)*.

$$\begin{aligned}
\max \sum_{j \in N} y_j - \frac{1}{2}\|v_0\|^2 & \\
y_j \leq \sum_{e \in i} w_{ej}(a_e \, w_{ej} + b_e) - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij} \rangle & \qquad \forall j \in N, \forall i \in \mathcal{S}_j \\
\langle v_{ij}, v_{i'k} \rangle \leq 2 \sum_{e \in i \cap i'} a_e \, w_{ej} \, w_{ek} & \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in N
\end{aligned}$$

Theorem 5.6.1. *For any instance of the above game, and any Nash equilibrium x, there exists a feasible (SDP-CG) solution with objective value at least $2/(3+\sqrt{5})\,C(x)$.*

Proof:
The vectors of the SDP will live in the space $\mathbb{R}^E$. Let $\alpha, \beta \geq 0$ be defined as in Lemma 5.4.3. We now state the dual fitting:

- $v_0(e) := -\beta \sqrt{a_e} \, \ell_e(x)$
- $v_{ij}(e) := \alpha \sqrt{a_e} \, w_{ej} \, \mathbb{1}_{\{e \in i\}} \qquad \forall j \in N, i \in \mathcal{S}_j$
- $y_j = \alpha\beta \, C_j(x) \qquad \forall j \in N$

Let us now compute the different inner products and norms that we need.

- $\|v_0\|^2 = \beta^2 \, \sum_{e \in E} \, a_e \, \ell_e(x)^2 \;\; \leq \beta^2 \, C(x)$
- $\|v_{ij}\|^2 = \alpha^2 \, \sum_{e \in i} a_e \, w_{ej}^2$

- $\langle v_0, v_{ij}\rangle = -\alpha\beta \sum_{e\in i} a_e\, w_{ej}\, \ell_e(x)$
- $\langle v_{ij}, v_{i'k}\rangle = \alpha^2 \sum_{e\in i\cap i'} a_e\, w_{ej}\, w_{ek}$

Let us now check feasibility of the solution. The second set of constraints is satisfied by the fourth computation above and the fact that $\alpha^2 = 2/\sqrt{5} \leq 2$. The first set of constraints is satisfied due to the Nash conditions (5.6.2). Indeed, under the above fitting, for every $j \in N, i \in \mathcal{S}_j$, the first set of SDP constraints read:

$$\alpha\beta\, C_j(x) \leq (1-\alpha^2/2)\sum_{e\in i} a_e\, w_{ej}^2 + \sum_{e\in i} w_{ej}\, b_e + \alpha\beta \sum_{e\in i} a_e\, w_{ej}\, \ell_e(x).$$

If there was a factor of $(1-\alpha^2/2) \leq 1$ multiplying the term $\sum_{e\in i} w_{ej}\, b_e$, then this would be equivalent to (5.6.2) because of the first condition of Lemma 5.4.3. Not having this term only increases the right hand side and thus ensures that this set of constraints is satisfied, implying that the SDP solution is feasible. The objective function can now be lower bounded as:

$$\sum_{j\in N} y_j - \frac{1}{2}\|v_0\|^2 \geq \alpha\beta \sum_{j\in N} C_j(x) - \frac{\beta^2}{2}\, C(x) = \left(\alpha\beta - \frac{\beta^2}{2}\right) C(x) = \frac{2}{3+\sqrt{5}}\, C(x)$$

where the last equality follows by the second property of Lemma 5.4.3. □

5.7 Recovering the Kawaguchi-Kyan bound for $P||\sum w_jC_j$

In this section, we show that we can recover the optimal bound of $(1+\sqrt{2})/2$ for the pure price of anarchy of the scheduling game on parallel machines $P||\sum w_jC_j$, where each machine uses increasing Smith ratios to schedule the jobs. To do so, we make use of a sequence of reductions to worst-case instances provided in [Sch11]. The first assumption that we can make is that $w_j = p_j$ for every job j. The *(SDP-SR)* dual semidefinite program shown in (5.4.4) and used in Section 5.5 for $R||\sum w_jC_j$ in this special case becomes the following. We denote the set of jobs by J and the set of machines by M.

$$\begin{aligned}
\max \sum_{j\in J} y_j - \frac{1}{2}\|v_0\|^2 & \\
y_j \leq p_j^2 - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle & \qquad \forall j \in J, \forall i \in M \\
\langle v_{ij}, v_{i'k}\rangle \leq p_j\, p_k\, \mathbb{1}_{\{i=i'\}} & \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in J
\end{aligned}$$

Moreover, the reduction in [Sch11] states that we may assume the instance only has two different processing times $\varepsilon, p > 0$, where ε is an arbitrarily small constant. Jobs with processing time ε are called small jobs, and the total workload of these jobs is $|M|$, i.e. the total number of small jobs is $|M|/\varepsilon$. Jobs with processing times p are called large jobs and the total number of large jobs is $k < |M|$, i.e. strictly less than the number of machines. In addition, in a pure Nash equilibrium x:

- All small jobs are started and completed in the interval $[0, 1]$.
- All large jobs are started at 1.

In this reduced instance, it is also possible to get an exact expression for the optimum solution. In particular, define $\alpha := m/(m-k)$ and $\beta := (m+pk)/m$, an optimal solution x^* then has cost:

$$C(x^*) = \begin{cases} kp^2 + \frac{m-k}{2}\alpha^2 & \text{if } p \geq \alpha \\ \frac{1}{2}\Big(kp^2 + m\beta^2\Big) & \text{if } p \leq \alpha \end{cases}$$

It can then be shown that in both cases $C(x)/C(x^*) \leq (1+\sqrt{2})/2$ through a simple calculus analysis. The reader is referred to [Sch11] for details.

We show here that we can construct a feasible dual solution to the SDP matching the objective value of $C(x^*)$, showing that the SDP does not have an integrality gap on such a reduced instance and thus implying that the price of anarchy is at most $(1+\sqrt{2})/2$ by a dual fitting proof.

Theorem 5.7.1. *For any instance of the above game on the reduced instance, there exists a feasible (SDP-SR) solution with objective value $C(x^*)$, implying that the price of anarchy is at most $(1+\sqrt{2})/2$.*

Proof:
The vectors in our dual fitting will live in the space $\mathbb{R}^M$. Let us denote the total number of machines by $m = |M|$, and let us set $\alpha := m/(m-k)$. We denote by $\mathbb{1}$ the all ones vector and by e_i the i^{th} standard basis vector.

We now state the dual fitting for the case where $p \geq \alpha$:

- $v_0 = -\alpha\,\mathbb{1}$
- If j is a large job, then set $v_{ij} = \alpha\, e_i$ and $y_j = p^2 + \alpha^2/2$
- If j is a small job, then set $v_{ij} = \varepsilon\, e_i$ and $y_j = \varepsilon\alpha$

Let us check that this solution is indeed feasible. Clearly, if $i \neq i'$, then $\langle v_{ij}, v_{i'k}\rangle = 0$ by orthogonality of e_i and $e_{i'}$. For two jobs $j \neq k$, we have that $\langle v_{ij}, v_{ik}\rangle \leq \|v_{ij}\|\,\|v_{ik}\| \leq p_j p_k$ where we use Cauchy-Schwarz for the first inequality and the

fact that $\alpha \leq p$ if some job is large for the second inequality. This shows that the second set of SDP constraints is satisfied.

Moreover, the first set of constraints is satisfied as well, as the SDP inequalities yield $y_j \leq p^2 + \alpha^2/2$ for large jobs and $y_j \leq \varepsilon^2/2 + \varepsilon\alpha$ for small jobs, which is satified by our choice of y_j. The objective value of this dual solution is then:

$$\sum_{j \in J} y_j - \frac{1}{2}\|v_0\|^2 = k\left(p^2 + \frac{\alpha^2}{2}\right) + \frac{m}{\varepsilon}\varepsilon\alpha - \frac{1}{2}m\alpha^2 = kp^2 + \frac{m-k}{2}\alpha^2$$

where the first equality follows since the number of small jobs is m/ε and the last equality follows by observing that $m\alpha = (m-k)\alpha^2$ by definition of α.

For the case where $p \leq \alpha$, we define $\beta := (m + pk)/m$. We now state the dual fitting:

- $v_0 = -\beta\,\mathbb{1}$
- If j is a large job, then set $v_{ij} = p\,e_i$ and $y_j = p^2/2 + \beta\,p$
- If j is a small job, then set $v_{ij} = \varepsilon\,e_i$ and $y_j = \varepsilon\beta$

Similarly to before, the second set of constraints is satisfied by orthogonality of the standard basis vectors and the fact that $\|v_{ij}\| = p_j$ for all jobs (either small or large). The first set of constraints yields $y_j \leq p^2/2 + \beta p$ for large jobs and $y_j \leq \varepsilon^2/2 + \varepsilon\beta$ for small jobs, which is clearly satisfied by our choice of y_j. The objective value of this dual solution is then:

$$\begin{aligned}\sum_{j \in J} y_j - \frac{1}{2}\|v_0\|^2 &= k\left(\frac{p^2}{2} + \beta p\right) + \frac{m}{\varepsilon}\varepsilon\beta - \frac{m\beta^2}{2} \\ &= \frac{kp^2}{2} + \beta(kp + m) - \frac{m\beta^2}{2} = \frac{1}{2}(kp^2 + m\beta^2)\end{aligned}$$

where the last equality follows by observing that $m\beta^2 = (m + pk)\beta$ by the definition of β. □

5.8 Robust price of anarchy

In this section, we describe how our proofs can be adapted to give bounds on the *coarse-correlated price of anarchy*, meaning that we can now generalize our results by considering *coarse-correlated equilibria*, instead of *mixed* (or *pure*) Nash equilibria.

Let N be a game with a strategy set $\mathcal{S}_j$ and payoff function C_j for every player $j \in N$. A distribution σ over $\mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ is a coarse correlated equilibrium if

$$\mathbb{E}_{X \sim \sigma}[C_j(X)] \leq \mathbb{E}_{X \sim \sigma}[C_j(X_{-j}, i)] \qquad \forall j \in N, i \in \mathcal{S}_j. \tag{5.8.1}$$

Note that this generalizes a mixed Nash equilibrium. In that case, σ is a product distribution, i.e. every player j picks a random strategy independently from its own distribution, which we denoted by $(x_{ij})_{i \in \mathcal{S}_j}$ previously. We note that our formulas for $C_j(x)$ - see for instance (5.4.1) - for non-binary x (i.e. interpreting x as a collection of probability distributions rather than an integer assignment) implicitly use this independence assumption, meaning that the current proofs do not directly go through for coarse-correlated equilibria. Let us first rewrite *(SDP-C)* (5.3.2) in a more convenient matrix form for this argument.

$$\begin{aligned} \max \sum_{j \in N} \varphi_j - \frac{1}{2} Y_{\{0,0\}} & \\ \varphi_j \leq C_{\{ij,\, ij\}} - \frac{1}{2} Y_{\{ij,ij\}} - Y_{\{0,ij\}} & \qquad \forall j \in N, i \in \mathcal{S}_j \\ Y_{\{ij,i'k\}} \leq 2\, C_{\{ij,\, i'k\}} & \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in N \\ Y \succeq 0 & \end{aligned}$$

One way to generalize our results is to consider random dual *(SDP-C)* solutions, i.e. doing a dual fitting on a realization $X \sim \sigma$, which induces binary random variables $\{X_{ij}\}_{j \in N, i \in \mathcal{S}_j}$ and $\{Z_{ej}\}_{j \in N, e \in E}$. For any price of anarchy dual fitting argument in this chapter, first replace every occurence of respectively x_{ij} and z_{ej} by X_{ij} and Z_{ej}, in which case v_0 and every y_j become random variables (note that every v_{ij} is always deterministic). To get a feasible dual solution, we now set $Y_{a,b} := \mathbb{E}_{X \sim \sigma}[\langle v_a, v_b \rangle]$ for every indices a, b as well as $\varphi_j := \mathbb{E}_{X \sim \sigma}[\, y_j \,]$.

The second set of constraints of *(SDP-C)* is always satisfied deterministically in our fittings, while the first set of constraints is satisfied by considering expectations, due to inequality (5.8.1). Moreover, Y is positive semidefinite since it is a convex combination of positive semidefinite matrices.

We thus get a feasible solution with objective value V satisfying

$$V \geq \rho\, \mathbb{E}_{X \sim \sigma}[C(X)]$$

for some desired bound $\rho \in [0, 1]$. Since the dual solution is feasible, we have $V \leq C(x^*)$, where x^* is the social optimum. Combining these two equations gives:

$$\mathbb{E}_{X \sim \sigma}[C(X)] \leq \frac{1}{\rho} C(x^*)$$

hence yielding a bound on the coarse-correlated price of anarchy.

5.9 Computation of the dual SDPs

5.9.1 Taking the dual

Recall that our primal semi-definite programming relaxation is the following.

$$\begin{aligned}
&\min\langle C, X\rangle \\
&\qquad \sum_{i\in\mathcal{S}_j} X_{\{ij,\,ij\}} = 1 && \forall j \in N \\
&\qquad\qquad X_{\{0,0\}} = 1 \\
&\qquad\qquad X_{\{0,\,ij\}} = X_{\{ij,\,ij\}} && \forall j \in N, i \in \mathcal{S}_j \\
&\qquad\qquad X_{\{ij,\,i'k\}} \geq 0 && \forall (i,j),(i',k) \text{ with } j,k > 0 \\
&\qquad\qquad X \succeq 0
\end{aligned}$$

It can be easily checked that the following form of semidefinite programs is a primal-dual pair. The dual variables $(\lambda_i)_i$ and $(\mu_j)_j$ respectively correspond to the equality and inequality constraints, whereas the matrix variable Y corresponds to the semidefinite constraint.

$$\begin{aligned}
&\min\langle C, X\rangle && \max \sum_i b_i \lambda_i \\
&\quad \langle A_i, X\rangle = b_i \quad \forall i && \\
&\quad \langle B_j, X\rangle \geq 0 \quad \forall j && \qquad Y = C - \sum_i \lambda_i A_i - \sum_j \mu_j B_j \\
&\qquad X \succeq 0 && \\
& && \qquad Y \succeq 0, \quad \mu \geq 0
\end{aligned}$$

Observe that our above primal SDP is in fact of that form. Let us denote by $(y_j)_{j\in N}, z$ and $(\sigma_{ij})_{j\in N, i\in\mathcal{S}_j}$ the dual variables respectively corresponding to the three sets of equality constraints. Let us denote by $\mu_{\{ij,i'k\}} \geq 0$ the dual variables corresponding to the inequality (or non-negativity) constraints. The dual objective then becomes $\sum_{j\in N} y_j + z$.

All the games considered will satisfy the fact that the objective matrix is all zeros in the first row and column: $C_{\{0,0\}} = 0$ and $C_{\{0,ij\}} = 0$ for every $j \in N$ and $i \in \mathcal{S}_j$. The dual matrix equality then becomes:

$$\begin{aligned}
Y_{\{0,0\}} &= -z \\
Y_{\{0,\,ij\}} &= \frac{\sigma_{ij}}{2} && \forall j \in N, i \in \mathcal{S}_j \\
Y_{\{ij,\,ij\}} &= C_{\{ij,\,ij\}} - y_j - \sigma_{ij} - \mu_{\{,\}} && \forall j \in N, i \in \mathcal{S}_j \\
Y_{\{ij,\,i'k\}} &= C_{\{ij,\,i'k\}} - \mu_{\{ij,\,i'k\}} && \forall (i,j) \neq (i',k) \text{ with } j,k > 0
\end{aligned}$$

Note that we can now eliminate the dual variables z and σ by the first two equalities. Moreover, we can eliminate the $\mu \geq 0$ variables by replacing the last

two equalities by inequalities. Let us now do the change of variable $Y' = 2Y$ and let the vectors of the Cholesky decomposition of Y' be v_0 and $(v_{ij})_{j\in N, i\in\mathcal{S}_j}$, meaning that $Y'_{a,\,b} = \langle v_a, v_b\rangle$ holds for all the entries of Y'. The dual SDP in vector form can then be rewritten as:

$$\begin{aligned}
\max \sum_{j\in N} y_j - \frac{1}{2}\|v_0\|^2 & \\
y_j \le C_{\{ij,\,ij\}} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle & \qquad \forall j\in N, i\in\mathcal{S}_j \\
\langle v_{ij}, v_{i'k}\rangle \le 2\, C_{\{ij,\,i'k\}} & \qquad \forall (i,j)\neq(i',k) \text{ with } j,k>0
\end{aligned}$$

5.9.2 Specializing it to the different games considered

Let us now describe how the objective matrix C looks like for the different games that we need. Recall from Section 5.3.1 that we need to pick a symmetric matrix C such that $C(x) = \langle C, X\rangle = \mathrm{Tr}(C^T X)$ where $X = (1,x)(1,x)^T$ is a binary rank one matrix and $C(x)$ is the social cost. By definition of the trace inner product, this is equivalent to:

$$C(x) = C_{\{0,0\}} + 2\sum_{j\in N, i\in\mathcal{S}_j} C_{\{0,ij\}}\, x_{ij} + \sum_{\substack{j,k\in N\\ i\in\mathcal{S}_j, i'\in\mathcal{S}_k}} C_{\{ij,\,i'k\}} x_{ij}\, x_{i'k}.$$

Recall also that $x_{ij}^2 = x_{ij}$ since $x_{ij}\in\{0,1\}$. Hence, if the social cost does not have constant terms, we will always be able to pick C such that $C_{\{0,0\}} = 0$ and $C_{\{0,ij\}} = 0$ for every $j\in N, i\in\mathcal{S}_j$, which we do for all the games below.

For the congestion game under the *Smith Rule* policy, the social cost in (5.4.2) can be written as:

$$C(x) = \sum_{j\in N} w_j\, C_j(x) = \sum_{\substack{j\in N\\ i\in\mathcal{S}_j\\ e\in i}} w_j\, p_{ej}\, x_{ij} + \frac{1}{2}\sum_{\substack{j\in N, k\neq j\\ i\in\mathcal{S}_j, i'\in\mathcal{S}_k\\ e\in i\cap i'}} w_j\, w_k \min\{\delta_{ej},\delta_{ek}\}\, x_{ij}\, x_{i'k}.$$

Therefore, the objective matrix C is the following:

$$C_{\{ij,\,ij\}} = \sum_{e\in i} w_j\, p_{ej} \quad , \quad C_{\{ij,\,i'k\}} = \frac{1}{2}\sum_{e\in i\cap i'} w_j\, w_k\, \min\{\delta_{ej},\delta_{ek}\}.$$

If one considers the scheduling problem $R||\sum w_jC_j$ under *Smith's Rule*, which is a special case of the previous setting, then

$$C_{\{ij,\,ij\}} = w_j\, p_{ij} \quad , \quad C_{\{ij,\,i'k\}} = \frac{1}{2}\, w_j\, w_k\, \min\{\delta_{ij},\delta_{ik}\}\, \mathbb{1}_{\{i=i'\}}.$$

For the congestion game under the *Rand* policy, the social cost in (5.4.11) gives

$$C_{\{ij,\,ij\}} = \sum_{e\in i} w_j\, p_{ej} \quad , \quad C_{\{ij,\,i'k\}} = \sum_{e\in i\cap i'} w_j\, w_k\, \frac{\delta_{ej}\delta_{ek}}{\delta_{ej}+\delta_{ek}}.$$

For the weighted affine congestion game, we have seen that

$$C(x) := \sum_{j \in N} C_j(x) = \sum_{e \in E} a_e \, \ell_e(x)^2 + b_e \, \ell_e(x)$$

where $\ell_e(x) = \sum_{j \in N} w_{ej} \sum_{i \in \mathcal{S}_j \colon e \in i} x_{ij}$. The objective matrix in that case is

$$C_{\{ij,\, ij\}} = \sum_{e \in i} w_{ej}(a_e \, w_{ej} + b_e) \quad , \quad C_{\{ij,\, i'k\}} = \sum_{e \in i \cap i'} a_e \, w_{ej} \, w_{ek}.$$

Chapter 6

Online load balancing via vector fitting

In this chapter, we show how our SDP dual fitting technique can be adapted to tightly bound the competitive ratio of online algorithms for two different online scheduling problems: online load balancing on unrelated machines and online $R||\sum w_j C_j$ with an optimal ordering on each machine. We recover the best known competitive ratios for deterministic and randomized algorithms for the first model. We also show an improved fractional algorithm, along with a matching lower bound. For the second model, we provide a deterministic and a randomized algorithm achieving a competitive ratio of 4, and show that this is optimal by presenting a matching lower bound.

6.1 Introduction

Scheduling a set of jobs over a collection of machines to optimize some objective function is one of the most important research topics in computer science theory and practice. In many practical scenarios, decisions must be made without complete knowledge of future events. This motivates the study of online scheduling, where jobs arrive over time and must be assigned to machines irrevocably upon arrival, without knowledge of future job characteristics.

In this work, we provide an SDP dual fitting framework to analyze online scheduling problems whose offline optimal solution can be modeled as a quadratic integer program. All our algorithms are analyzed in a unified way, by making a set of constraints correspond to an equilibrium condition satisfied by the online algorithm at every time step. Similar approaches were developed in [AGK12, JLM25, IKMP14, GGKS19, GMUX20, IKM17, Jäg23, LM22, GKP12] by using convex programs or time-indexed linear programs. However, to the best of our knowledge, this is the first time semidefinite programs are used for such arguments.

We apply our technique to two such problems. The first one concerns the load balancing problem on unrelated machines, where the objective is to minimize the square of the L_2 norm of the loads of the machines. This problem has

originally been studied in [AAG+95], who showed that the greedy algorithm is $3+2\sqrt{2} \approx 5.828$-competitive. This algorithm is in fact *optimal* among deterministic algorithms [Car08]. Improving on the greedy strategy by using randomization was a challenging open question for more than a decade until a 5-competitive randomized algorithm was shown by [Car08], and this is currently the best known algorithm to date.

We then study the problem of minimizing the sum of weighted completion times on unrelated machines, denoted as $R||\sum w_j C_j$. In the offline setting, this is in fact only an assignment problem, since the optimal ordering on every machine is to schedule the jobs according to increasing Smith ratios, defined as the ratio between the processing time and the weight of a job. We consider an online model where jobs arrive one by one and need to irrevocably be assigned to a machine upon arrival. When assigned, a job then enters the schedule in the right position with respect to the Smith ratio. Equivalently, the arrival order is an online decision, but the ordering on each machine can be made offline. This model has previously been studied in [GMUX20].

6.1.1 Our contributions

Our main contribution of this chapter is to extend the unified dual fitting approach to online scheduling problems. The Nash equilibria or local optima inequalities of the previous chapter are now replaced by inequalities satisfied by an online algorithm at every time step. Using this structure, we are able to obtain numerous results in a unified way. To the best of our knowledge, this is the first time such SDP dual fitting arguments are used in online algorithms.

We first study the online load balancing problem on unrelated machines, where the goal is to minimize the sum of squares of the loads. We first analyze the greedy algorithm for a generalized model where each job can be assigned to hyperedges of machines and show that it is $(3+2\sqrt{2}) \approx 5.828$-competitive, generalizing a result of [AAG+95]. The greedy algorithm is known to be optimal among deterministic algorithms, however an improved 5-competitive randomized algorithm has been shown by [Car08]. We show how to recover this result through our approach, as well as providing a matching lower bound for any *independent* randomized rounding algorithm. Finally, we provide an *optimal* 4-competitive fractional algorithm, along with a matching lower bound.

We then study an online model for the scheduling problem $R||\sum w_j C_j$. Each job arrives online and needs to be assigned to a machine at its arrival by an online algorithm. When a job is assigned to a machine, it enters the position in the schedule at the right position with respect to the Smith ratio. Equivalently, the ordering of the jobs on each machine is an offline decision. It is shown in [GMUX20] that the greedy algorithm is 4-competitive. We show that this greedy algorithm can be analyzed through our approach in a more general hypergraph model and provide an alternative randomized algorithm in the standard model

achieving the same bound of 4. We also show a matching lower bound, even against fractional algorithms.

6.1.2 Further related work

In the offline setting, the unrelated machine scheduling problem $R||\sum w_j C_j$ is APX-hard [HSW98]. Constant factor approximation algorithms are however possible, with a simple 3/2-approximation achievable by rounding a convex relaxation [Sku01, SS99]. A $3/2 - c$ approximation for some absolute constant $c > 0$ has been shown in [BSS16]. Building on this, subsequent improvements have been made [IS20, IL23, Har24] with the current best (to the best of our knowledge) approximation algorithm for this problem obtaining a ratio of $1.36+\varepsilon$ [Li24]. In the special case where Smith ratios are uniform, an improved bound of $(1+\sqrt{2})/2+\varepsilon$ is known [KST17].

The unrelated load balancing problem to minimize the square of the L_2 norm admits an easy 2-approximation by rounding a convex program [AE05]. The best known approximation algorithm obtains a bound of 4/3 and is given in [IL23] by using a time-indexed LP and the Shmoys-Tardos rounding algorithm [ST93]. For the more general L_p norm with $p < \infty$, [AAG+95] show how to get a $\Theta(p)$ approximation. In a breakthrough, [AE05] improved this to a 2-approximation by using (again) the Shmoys-Tardos rounding algorithm. Further improvements have been made in [KMPS09] by a new dependent rounding approach. The L_∞ norm corresponds to the makespan minimization problem. It is known to be NP-hard to approximate within a factor of 1.5, and a 2-approximation is given in the classic result of [ST93].

In the online unrelated setting to minimize the square of the L_2 norm, [AAG+95] show that the greedy algorithm is $3 + 2\sqrt{2} \approx 5.828$-competitive. This was shown to be a tight bound in [CFK+06] even in the restricted identical machines setting, and this bound is even best possible among deterministic algorithms [Car08]. Improving this bound was an open question for more than a decade until a 5-competitive randomized algorithm was shown by [Car08]. In fact, this approach improved the best known bounds for the more general L_p norm for many values of $p > 0$. Improvements have been made for the unit weight setting on related machines [STZ04, CFK+06]. In particular, the greedy algorithm is ≈ 4.06-competitive on restricted parallel machines under unit weights [CFK+06].

Many other models of online scheduling of jobs on machines have been studied. One natural model consists of jobs having a release date and arriving online at that point in time, where the objective is to minimize an objective function depending on the weighted flow time of jobs. In this model, strong lower bounds are known, even for preemptive algorithms [KTW96, GK07, CKZ01]. Given these lower bounds, such scheduling problems have been considered in the speed augmentation model, where each machine is allowed to run at a ε-fraction faster speed than the offline optimum [CGKM09, IM11, KP00a, BP03]. Dual fitting

approaches on LPs and convex programs for different scheduling problems have been developed in [AGK12, JLM25, IKMP14, GGKS19, GMUX20, IKM17, Jäg23, LM22, GKP12].

6.1.3 Outline of the chapter

In Section 6.2, we formally introduce the two online problems studied. We then study the first model in Section 6.3 and the second model in Section 6.4.

6.2 Preliminaries

6.2.1 Problems studied

Online unrelated load balancing. A set of resources E is given. A set of jobs N arrives online in an adversarial order. Each time a job $j \in N$ arrives, it reveals a subset $\mathcal{S}_j \subset E$ of resources it can be assigned to with unrelated weights $w_{ij} \geq 0$ associated with every $i \in \mathcal{S}_j$. An online integral algorithm needs to irrevocably pick a resource $i \in \mathcal{S}_j$ to assign that job to. We denote by $x_{ij} \in \{0, 1\}$ the indicator variable of whether the algorithm assigns j to i. The load of a resource $i \in E$ is the total amount of weight assigned to it and is denoted as:

$$L_i(x) = \sum_{j \in N} w_{ij}\, x_{ij}.$$

Since jobs arrive online, we order them as $N = \{1, \dots n\}$, where job j arrives before job k if $j < k$. For each resource $i \in E$, we denote the load over time as

$$L_i^{(j)}(x) = \sum_{k \leq j} w_{ik}\, x_{ik} \qquad \forall j \in N.$$

In words, this is the load of a resource after job j arrived. Observe that the final load is $L_i(x) = L_i^{(n)}(x)$ and we define $L_i^{(0)} := 0$ for convenience. The goal of the problem is to minimize the following objective function, which is the sum of squares of the loads:

$$C(x) = \sum_{i \in E} L_i(x)^2.$$

Online unrelated weighted completion time. A set of resources E is given. A set of jobs N arrives online in an adversarial order. Each time a job $j \in N$ arrives, it reveals a weight $w_j \geq 0$, and a subset $\mathcal{S}_j \subset E$ of resources it can be assigned to with unrelated processing times $p_{ij} \geq 0$ for every $i \in \mathcal{S}_j$. Once every job has arrived, every resource reorders the jobs assigned to it by increasing Smith ratios $\delta_{ij} := p_{ij}/w_j$. We denote the order induced by this rule on each resource

by $k \prec_i j$ (whenever $\delta_{ik} \leq \delta_{ij}$), where ties are broken arbitrarily. The completion time of every job is defined as:

$$C_j(x) = \sum_{i \in \mathcal{S}_j} x_{ij} \Big(p_{ij} + \sum_{k \prec_i j} p_{ik}\, x_{ik} \Big).$$

The goal of the problem is to minimize the sum of weighted completion times:

$$C(x) := \sum_{j \in N} w_j\, C_j(x).$$

6.2.2 Hypergraph generalizations

We also consider the following generalizations of these models.

Load balancing. A generalization of this problem is when each $j \in N$ now has a collection $\mathcal{S}_j \subseteq 2^E$ of *subsets* or *hyperedges* of resources it can be assigned to. In that case, we denote by $x_{ij} \in \{0, 1\}$ the indicator variable of whether j is assigned to the hyperedge $i \in \mathcal{S}_j$ and $z_{ej} = \sum_{i \in S_j : e \in i} x_{ij}$ whether j is assigned to a hyperedge containing $e \in E$. The load of a resource e is still the total amount of weight assigned to it:

$$L_e(x) = \sum_{j \in N} w_{ej}\, z_{ej} = \sum_{j \in N} w_{ej} \sum_{i \in S_j : e \in i} x_{ij}.$$

The goal of the problem is again to minimize $\sum_{e \in E} L_e(x)^2$.

Smith's Rule. Similarly to above, each $j \in N$ now has a collection $\mathcal{S}_j \subseteq 2^E$ of *subsets* or *hyperedges* of resources it can be assigned to. We denote again $z_{ej} = \sum_{i \in S_j : e \in i} x_{ij}$ and the completion time of a job is now defined as follows:

$$C_j(x) = \sum_{i \in S_j} x_{ij} \sum_{e \in i} \Big(p_{ej} + \sum_{k \prec_e j} p_{ek}\, z_{ek} \Big).$$

The goal of the problem is still to minimize $\sum_{j \in N} w_j C_j(x)$.

6.3 Online load balancing on unrelated machines

6.3.1 The greedy algorithm

We now consider the following algorithm named Greedy for the hypergraph model. Whenever a job $j \in N$ arrives, Greedy picks $i \in \mathcal{S}_j$ (i.e. sets $x_{ij} = 1$) which gives the least increase in the global objective function. The key property of the greedy algorithm is the following lemma.

Algorithm 6.3.1 Greedy algorithm

when $j \in N$ **arrives:**
Set $x_{ij} = 1$ for $i \in \mathcal{S}_j$ giving the minimal increase in the objective function
return x

Lemma 6.3.1. *For any solution $(x_{ij})_{j\in N, i\in\mathcal{S}_j}$ constructed by* Greedy*, the following inequalities are satisfied:*

$$\sum_{e\in E}\left(L_e^{(j)}(x)\right)^2 - \sum_{e\in E}\left(L_e^{(j-1)}(x)\right)^2 \leq \sum_{e\in i}\left(w_{ej}^2 + 2\, L_e^{(j-1)}(x)\, w_{ej}\right) \qquad \forall j \in N, \forall i \in \mathcal{S}_j.$$

Proof:
Let us fix an arbitrary $j \in N$. By definition of the greedy algorithm, whenever $j \in N$ arrives, one has the following inequality:

$$\sum_{e\in E}\left(L_e^{(j)}(x)\right)^2 \leq \sum_{e\in E}\left(L_e^{(j-1)}(x) + w_{ej}\mathbb{1}_{\{e\in i\}}\right)^2 \qquad \forall i \in \mathcal{S}_j.$$

Expanding out the right hand side and rearranging terms gives:

$$\sum_{e\in E}\left(L_e^{(j)}(x)\right)^2 - \sum_{e\in E}\left(L_e^{(j-1)}(x)\right)^2 \leq \sum_{e\in E}\left(w_{ej}^2\mathbb{1}_{\{e\in i\}} + 2\, L_e^{(j-1)}(x)\, w_{ej}\mathbb{1}_{\{e\in i\}}\right).$$

□

We will now analyze the competitive ratio of Greedy through a dual fitting argument on the following semidefinite programming relaxation that we call *(SDP-LB)*. This program is a specialization of (5.3.2).

$$\begin{aligned}
\max \sum_{j\in N} y_j - \frac{1}{2}\|v_0\|^2 & \\
y_j \leq \sum_{e\in i} w_{ej}^2 - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle & \qquad \forall j \in N, \forall i \in \mathcal{S}_j \\
\langle v_{ij}, v_{i'k}\rangle \leq 2\sum_{e\in i\cap i'} w_{ej}\, w_{ek} & \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in N
\end{aligned}$$

We will do the fitting in a way to ensure that the first set of constraints is satisfied due to the inequalities of Lemma 6.3.1. We first need two constants that will play a key role in the fitting. The first property will ensure feasibility of the solution, whereas the second one will be the constant in front of the objective function determining the competitive ratio.

Lemma 6.3.2. *Let $\alpha, \beta \geq 0$ be defined as $\alpha^2 = \sqrt{2}$ and $\beta = (2-\alpha^2)/\alpha = (\sqrt{2}-1)\alpha$. The following properties hold:*

- $1 - \alpha^2/2 = \alpha\beta/2$
- $(\alpha\beta - \beta^2)/2 = 1/(3 + 2\sqrt{2})$

Proof:
The first property is immediate by definition of β. The second property consists of simple computations and is omitted (it can also be checked on a computer). □

Theorem 6.3.3. *For any instance of the online load balancing problem, and any solution $(x_{ij})_{j \in N, i \in \mathcal{S}_j}$ obtained by* Greedy*, there exists a feasible (SDP-LB) solution with objective value at least*

$$\frac{1}{3 + 2\sqrt{2}} \sum_{e \in E} L_e(x)^2.$$

By weak duality, this implies that the competitive ratio of Greedy *is at most* $3 + 2\sqrt{2} \approx 5.828$.

Remark 6.3.4. This generalizes the result given in [AAG+95] for the "standard" online weighted load balancing problem. Moreover, this bound is tight with a matching lower bound given in [CFK+06] for restricted identical machines. In fact, it even turns out that this algorithm is best possible among deterministic (integral) algorithms [Car08].

Proof:
The vectors of the SDP will live in the space $\mathbb{R}^E$. Let $\alpha, \beta \geq 0$ be defined as in Lemma 6.3.2. We now state the dual fitting:

- $v_0(e) := -\beta \, L_e(x)$
- $v_{ij}(e) := \alpha \, w_{ej} \, \mathbb{1}_{\{e \in i\}}$ $\qquad \forall j \in N, i \in \mathcal{S}_j$
- $y_j := \frac{\alpha\beta}{2} \left[\sum_{e \in E} \left(L_e^{(j)}(x) \right)^2 - \sum_{e \in E} \left(L_e^{(j-1)}(x) \right)^2 \right]$ $\qquad \forall j \in N$

Let us now compute the different inner products and norms that we need.

- $\|v_0\|^2 = \beta^2 \, \sum_{e \in E} \, L_e(x)^2$
- $\|v_{ij}\|^2 = \alpha^2 \, \sum_{e \in i} \, w_{ej}^2$
- $\langle v_0, v_{ij} \rangle = -\alpha\beta \, \sum_{e \in i} \, w_{ej} \, L_e(x)$
- $\langle v_{ij}, v_{i'k} \rangle = \alpha^2 \, \sum_{e \in i \cap i'} \, w_{ej} \, w_{ek}$

Let us now check feasibility of the solution. The second set of constraints is satisfied by the fourth computation above and the fact that $\alpha^2 \leq 2$. The first set of constraints turns out to be satisfied due to the inequalities valid for GREEDY stated in Lemma 6.3.2. Indeed, under the above fitting, for every $j \in N, i \in \mathcal{S}_j$, the first set of SDP constraints reads:

$$y_j \leq \left(1 - \frac{\alpha^2}{2}\right) \sum_{e \in i} w_{ej}^2 + \frac{\alpha\beta}{2} \sum_{e \in i} 2\, w_{ej}\, L_e(x).$$

By our choice of fitting for y_j and the first property of Lemma 6.3.2, the constant terms cancel out on both sides of the inequality. These inequalities are then clearly satisfied by Lemma 6.3.1, since $L_e^{(j-1)}(x) \leq L_e(x)$. To argue about the objective function, observe that

$$\sum_{j \in N} y_j = \frac{\alpha\beta}{2} \sum_{j \in N} \Big[\sum_{e \in E} \left(L_e^{(j)}(x)\right)^2 - \sum_{e \in E} \left(L_e^{(j-1)}(x)\right)^2 \Big] = \frac{\alpha\beta}{2} \sum_{e \in E} L_e(x)^2$$

where the last equality follows from exchanging the summations, observing that the inner sum is telescoping and that $L_e^{(n)}(x) = L_e(x)$ is the final load on every resource. The objective function is therefore equal to:

$$\sum_{j \in N} y_j - \frac{1}{2}\|v_0\|^2 = \left(\frac{\alpha\beta}{2} - \frac{\beta^2}{2}\right) \sum_{e \in E} L_e(x)^2 = \frac{1}{3 + 2\sqrt{2}} \sum_{e \in E} L_e(x)^2$$

where the last equality follows by the second property of Lemma 6.3.2. □

6.3.2 An improved randomized algorithm

In this section, we show how to improve on the greedy algorithm in the standard model using randomization, yielding a 5-competitive algorithm. The algorithm we use is due to [Car08] and is called BALANCE. Whenever $j \in N$ arrives, we consider the following potential functions $f_{ij} : [0,1] \to \mathbb{R}$ for every $i \in \mathcal{S}_j$:

$$f_{ij}(t) := w_{ij}^2 + 4\, w_{ij} \left(\mathbb{E}\left[L_i^{(j-1)}(X)\right] + t\, w_{ij} \right). \tag{6.3.1}$$

The algorithm BALANCE then defines a probability distribution $(x_{ij})_{i \in S_j}$ which ensures that

$$x_{ej} > 0 \implies f_{ej}(x_{ej}) \leq f_{ij}(x_{ij}) \qquad \forall i \in \mathcal{S}_j.$$

For convenience, we also let $x_{ej} = 0$ if $e \notin \mathcal{S}_j$. Observe that this means that for every $e \in S_j$ with $x_{ej} > 0$, we get that $f_{ej}(x_{ej}) = \lambda$ for some constant λ, whereas $f_{ej}(x_{ej}) \geq \lambda$ if $x_{ej} = 0$. In particular, we also get the following inequality:

$$\sum_{e \in E} x_{ej}\, f_{ej}(x_{ej}) \leq f_{ij}(x_{ij}) \qquad \forall i \in \mathcal{S}_j. \tag{6.3.2}$$

Algorithm 6.3.2 Waterfilling and independent rounding algorithm

when $j \in N$ **arrives:**

Compute $(x_{ij})_{i \in \mathcal{S}_j}$ such that $\sum_i x_{ij} = 1$ and (6.3.2) holds
Assign j to $i \in \mathcal{S}_j$, i.e. set $X_{ij} = 1$ with probability x_{ij}

return X

This holds since the left hand side equals to λ, due to $\sum_{e \in E} x_{ej} = 1$. Balance can be seen as a waterfilling algorithm.

We show how to analyze this algorithm using our dual SDP framework, yielding a simpler proof of this result. We state the result here for the standard model, meaning that $\mathcal{S}_j \subseteq E$ for every $j \in N$. Observe that now, both indices $i \in \mathcal{S}_j$ and $e \in E$ refer to elements of E. We will use both interchangeably whenever convenient. In this model, the dual SDP becomes:

$$\begin{aligned} \max \sum_{j \in N} y_j - \frac{1}{2}\|v_0\|^2 & \\ y_j \le w_{ij}^2 - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle & \qquad \forall j \in N, \forall i \in \mathcal{S}_j \\ \langle v_{ij}, v_{i'k}\rangle \le 2\, w_{ij}\, w_{i'k}\, \mathbb{1}_{\{i=i'\}} & \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in N. \end{aligned}$$

Similarly to the greedy algorithm, we first need a lemma stating key inequalities satisfied by this algorithm. Let us first define

$$\begin{aligned} \delta_e^{(j)}(x) &:= \mathbb{E}\left[L_e^{(j)}(X)\right]^2 - \mathbb{E}\left[L_e^{(j-1)}(X)\right]^2, \\ \Delta_e^{(j)}(x) &:= \mathbb{E}\left[L_e^{(j)}(X)^2\right] - \mathbb{E}\left[L_e^{(j-1)}(X)^2\right]. \end{aligned} \tag{6.3.3}$$

The first quantity is the difference, after j has arrived, between the squares of the first moment of the load of a resource e, whereas the second one is the difference of the second moments. Let us now expand the definition of these terms. Observe that for every $e \in E$, we have $L_e^{(j)}(X) = L_e^{(j-1)}(X) + X_{ej}\, w_{ej}$, meaning that the squares of the random loads vary as follows:

$$L_e^{(j)}(X)^2 - L_e^{(j-1)}(X)^2 = X_{ej}\, w_{ej}^2 + 2\, L_e^{(j-1)}(X)\, X_{ej}\, w_{ej}$$

where we use the fact that $X_{ej}^2 = X_{ej}$, since it is a binary random variable. The term $\Delta_e^{(j)}(x)$ is simply the expectation of the above equation and can be written as follows:

$$\Delta_e^{(j)}(x) = x_{ej}\, w_{ej}^2 + 2\, \mathbb{E}[L_e^{(j-1)}(X)]\, x_{ej}\, w_{ej} \tag{6.3.4}$$

where we use that $\mathbb{E}[X_{ej}] = x_{ej}$ and the fact that Balance makes *independent* random choices for different jobs, meaning that the random variables $L_e^{(j-1)}(X)$

and X_{ej} are independent. In addition, we get:

$$\begin{aligned}\delta_e^{(j)}(x) &= \left(\mathbb{E}[L_e^{(j-1)}(X)] + x_{ej}\, w_{ej}\right)^2 - \left(\mathbb{E}[L_e^{(j-1)}(X)]\right)^2 \\ &= x_{ej}^2\, w_{ej}^2 + 2\,\mathbb{E}[L_e^{(j-1)}(X)]\, x_{ej}\, w_{ej}. \end{aligned} \tag{6.3.5}$$

Observe that the only difference between $\delta_e^{(j)}(x)$ and $\Delta_e^{(j)}(x)$ is the square for x_{ej} in the first term, which will crucially be exploited in the following lemma, which states key inequalities satisfied by Balance needed for the dual fitting.

Lemma 6.3.5. *For any solution $(x_{ij})_{j\in N, i\in\mathcal{S}_j}$ constructed by Balance, the following inequalities are satisfied for every $j \in N$:*

$$\sum_{e\in E}\left(\delta_e^{(j)}(x) + \Delta_e^{(j)}(x)\right) \le w_{ij}^2 + 4\, w_{ij}\,\mathbb{E}\left[L_i(X)\right] \qquad \forall i \in \mathcal{S}_j.$$

Proof:
We first compute

$$\delta_e^{(j)}(x) + \Delta_e^{(j)}(x) = x_{ej} w_{ej}\left(w_{ej} + w_{ej}x_{ej} + 4\,\mathbb{E}[L_e^{(j-1)}(X)]\right).$$

Observe now that $\mathbb{E}[L_e^{(j)}(X)] = \mathbb{E}[L_e^{(j-1)}(X)] + w_{ej}x_{ej}$, which means that we can upper bound the above by:

$$\begin{aligned}\delta_e^{(j)}(x) + \Delta_e^{(j)}(x) &\le x_{ej} w_{ej}\left(w_{ej} + 4\,\mathbb{E}[L_e^{(j)}(X)]\right) \\ &= x_{ej}\left(w_{ej}^2 + 4\, w_{ej}\,\mathbb{E}[L_e^{(j)}(X)]\right) = x_{ej}\, f_{ej}(x_{ej})\end{aligned}$$

where we use the definition (6.3.1) in the last equality. By the equilibrium condition (6.3.2), we get:

$$\sum_{e\in E}\left(\delta_e^{(j)}(x) + \Delta_e^{(j)}(x)\right) \le f_{ij}(x_{ij}) \quad \forall i \in \mathcal{S}_j.$$

Looking again at the definition (6.3.1) finishes the proof. □

We also need a lemma about the right constants for the dual fitting.

Lemma 6.3.6. *Let $\alpha, \beta \ge 0$ be defined as $\alpha = 2\sqrt{2/5} \approx 1.265$ and $\beta = \sqrt{2/5}$. The following properties hold:*

- $1 - \alpha^2/2 = \alpha\beta/4$
- $\alpha\beta/4 = 1/5$

Proof:
The proof is immediate. □

We are now ready to analyze Balance using our dual fitting approach.

Theorem 6.3.7. *For any instance of the online load balancing problem, and any solution $(x_{ij})_{j\in N, i\in\mathcal{S}_j}$ obtained by* Balance*, there exists a feasible (SDP-LB) solution with objective value at least*

$$\frac{1}{5}\sum_{e\in E}\mathbb{E}\left[L_e(X)^2\right].$$

By weak duality, this implies that the competitive ratio of Balance *is at most* 5.

Proof:
The vectors of the SDP will live in the space $\mathbb{R}^E$. Let $\alpha, \beta \geq 0$ be defined as in Lemma 6.3.6. We now state the dual fitting:

- $v_0(e) := -\beta\,\mathbb{E}\left[L_e(X)\right]$
- $v_{ij}(e) := \alpha\, w_{ej}\, \mathbb{1}_{\{e=i\}}$ $\qquad \forall j \in N, i \in \mathcal{S}_j$
- $y_j = \frac{1}{5}\sum_{e\in E}\left(\delta_e^{(j)}(x) + \Delta_e^{(j)}(x)\right)$ $\qquad \forall j \in N.$

Let us now compute the different inner products and norms that we need.

$$\|v_0\|^2 = \beta^2 \sum_{e\in E}\mathbb{E}[L_e(X)]^2 \qquad \|v_{ij}\|^2 = \alpha^2\, w_{ij}^2$$
$$\langle v_0, v_{ij}\rangle = -\alpha\beta\, w_{ij}\,\mathbb{E}[L_i(X)] \qquad \langle v_{ij}, v_{i'k}\rangle = \alpha^2\, w_{ij}\, w_{i'k}\, \mathbb{1}_{\{i=i'\}}.$$

The second set of constraints of the SDP is satisfied due to the last computation above and the fact that $\alpha^2 = 8/5 \leq 2$. The first set of constraints under the above fitting reads:

$$\begin{aligned} y_j &\leq w_{ij}^2 - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle \\ &\iff \frac{1}{5}\sum_{e\in E}\left(\delta_e^{(j)}(x) + \Delta_e^{(j)}(x)\right) \leq \left(1 - \frac{\alpha^2}{2}\right) w_{ij}^2 + \frac{\alpha\beta}{4}\, 4\, w_{ij}\,\mathbb{E}[L_i(X)]. \end{aligned}$$

Observe that $1 - \alpha^2/2 = \alpha\beta/4 = 1/5$ by Lemma 6.3.6, meaning that this set of constraints is now satisfied by Lemma 6.3.5. To argue about the objective function, observe that

$$\sum_{j\in N} y_j = \frac{1}{5}\sum_{e\in E}\sum_{j\in N}\left(\delta_e^{(j)}(x) + \Delta_e^{(j)}(x)\right) = \frac{1}{5}\sum_{e\in E}\left(\mathbb{E}\left[L_e(X)\right]^2 + \mathbb{E}\Big[L_e(X)^2\Big]\right)$$

by the definition of the moment increments in (6.3.3) and the fact that the sum is telescoping. Hence, the objective function is

$$\sum_{j\in N} y_j - \frac{1}{2}\|v_0\|^2 = \sum_{j\in N} y_j - \frac{\beta^2}{2}\sum_{e\in E}\mathbb{E}[L_e(X)]^2 = \frac{1}{5}\sum_{e\in E}\mathbb{E}\Big[L_e(X)^2\Big]$$

where the second equality follows from $\beta^2/2 = 1/5$ since the definition of β states $\beta = \sqrt{2/5}$. □

6.3.3 An optimal fractional algorithm

In this section, we show how to get a 4-competitive *optimal* fractional algorithm. The algorithm is still of waterfilling type but with the following potential functions $f_{ij} : [0,1] \to \mathbb{R}$ for every $i \in \mathcal{S}_j$:

$$f_{ij}(t) := 2\, w_{ij} \left(L_i^{(j-1)}(x) + t\, w_{ij} \right). \tag{6.3.6}$$

We call the following algorithm FracBalance.

Algorithm 6.3.3 Optimal fractional waterfilling algorithm

when $j \in N$ **arrives:**
Compute $(x_{ij})_{i \in \mathcal{S}_j}$ such that (6.3.2) holds for the new definition of f_{ij}
return x

At every time step $j \in N$, let us denote the increments as

$$\delta_e^{(j)}(x) = L_e^{(j)}(x)^2 - L_e^{(j-1)}(x)^2 \quad \forall e \in E.$$

This is in fact exactly the same formula as for the increment of the square of the first moment in the randomized integral case. However, the key difference here is that the total cost is now also determined by summing up these increments:

$$\sum_{e \in E} L_e(x)^2 = \sum_{e \in E} \sum_{j \in N} \delta_e^{(j)}(x).$$

Lemma 6.3.8. *For any solution $(x_{ij})_{j \in N, i \in \mathcal{S}_j}$ constructed by FracBalance, the following inequalities are satisfied for every $j \in N$:*

$$\sum_{e \in E} \delta_e^{(j)}(x) \le 2\, w_{ij}\, L_i(x) \qquad \forall i \in \mathcal{S}_j.$$

Proof:
Let us compute:

$$\begin{aligned} \delta_e^{(j)}(x) &= \left(L_e^{(j-1)}(x) + w_{ej} x_{ej} \right)^2 - L_e^{(j-1)}(x)^2 = x_{ej}\, w_{ej}\, \left(2\, L_e^{(j-1)}(x) + x_{ej} w_{ej} \right) \\ &\le x_{ej}\, f_{ej}(x_{ej}). \end{aligned}$$

By the equilibrium condition (6.3.2), we get:

$$\sum_{e \in E} \delta_e^{(j)}(x) \le f_{ij}(x_{ij}) \quad \forall i \in \mathcal{S}_j.$$

By definition of f_{ij} given in (6.3.6), we then get that for every $i \in \mathcal{S}_j$:

$$f_{ij}(x_{ij}) = 2\, w_{ij} \left(L_i^{(j-1)}(x) + x_{ij}\, w_{ij} \right) = 2\, w_{ij}\, L_i^{(j)}(x) \le 2\, w_{ij}\, L_i(x).$$

□

Theorem 6.3.9. *For any instance and any solution $(x_{ij})_{j\in N, i\in\mathcal{S}_j}$ obtained by* FracBalance*, there exists a feasible (SDP-LB) solution with objective value at least*

$$\frac{1}{4}\sum_{e\in E} L_e(x)^2.$$

By weak duality, this implies that the competitive ratio of FracBalance *is at most* 4*.*

Proof:
The vectors of the SDP will live in the space $\mathbb{R}^E$. Let $\alpha = \sqrt{2}$ and $\beta = 1/\sqrt{2}$. We now state the dual fitting:

- $v_0(e) := -\beta\, L_e(x)$
- $v_{ij}(e) := \alpha\, w_{ej}\, \mathbb{1}_{\{e=i\}}$ $\qquad \forall j \in N, i \in \mathcal{S}_j$
- $y_j = \frac{1}{2}\, \sum_{e\in E} \delta_e^{(j)}(x)$ $\qquad \forall j \in N$

Let us now compute the different inner products and norms that we need.

$$\|v_0\|^2 = \frac{1}{2}\sum_{e\in E} L_e(x)^2 \qquad \|v_{ij}\|^2 = 2\, w_{ij}^2$$
$$\langle v_0, v_{ij}\rangle = -w_{ij}\, L_i(x) \qquad \langle v_{ij}, v_{i'k}\rangle = 2\, w_{ij}\, w_{i'k}\, \mathbb{1}_{\{i=i'\}}.$$

The second set of constraints of the SDP is satisfied due to the last computation above. The first set of constraints under the above fitting reads:

$$y_j \le w_{ij}^2 - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle \iff \frac{1}{2}\sum_{e\in E}\delta_e^{(j)}(x) \le w_{ij}\, L_i(x).$$

These are clearly satisfied by Lemma 6.3.8. The objective function now becomes:

$$\sum_{j\in N} y_j - \frac{1}{2}\|v_0\|^2 = \frac{1}{2}\sum_{e\in E} L_e(x)^2 - \frac{1}{4}\sum_{e\in E} L_e(x)^2 = \frac{1}{4}\sum_{e\in E} L_e(x)^2.$$

□

6.3.4 A lower bound for fractional algorithms

In this section, we show that our fractional algorithm is optimal by providing a matching lower bound against any fractional algorithm.

Theorem 6.3.10. *For any $\varepsilon > 0$, there exists an online instance to the online unrelated load balancing problem such that any fractional algorithm has a competitive ratio of at least $4 - \varepsilon$.*

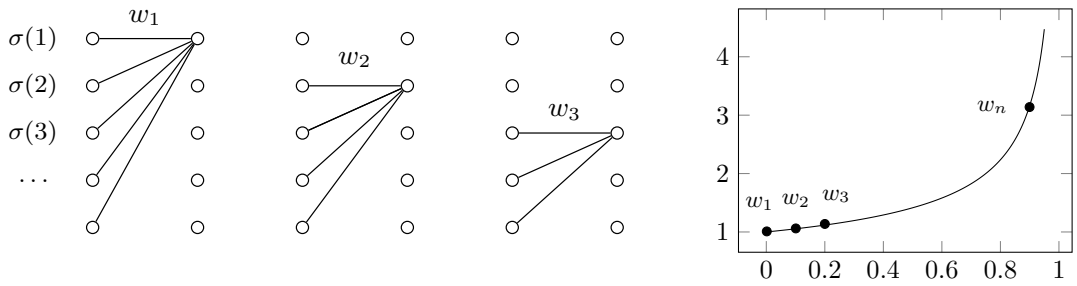


Figure 6.1: Illustration of the instance and of the weight distribution (for $n = 10$).

Proof:
Let $n \in \mathbb{N}$, the instance will satisfy $E = N = [n]$. Consider a uniformly at random permutation $\sigma : [n] \to [n]$ of the resources. The feasible resources for every job j will now be:

$$\mathcal{S}_j = \{\sigma(i) : j \leq i \leq n\} \qquad \forall j \in [n].$$

Consider the function $f : [0, 1] \to \mathbb{R}$ defined as $f(x) = 1/\sqrt{1-x}$. The weights of the instance are then defined as

$$w_{ej} = w_j = f\left(\frac{j-1}{n}\right) \qquad \forall j \in N, \forall e \in \mathcal{S}_j.$$

Observe that the weights do not depend on the resource/machine, meaning that the lower bound will hold even in the restricted identical machines setting. To simplify computations, we will below often use the following lower and upper approximations of a sum by an integral for an arbitrary increasing function $g : [0, 1] \to \mathbb{R}$ and any $k \leq n$:

$$\int_0^{(k-1)/n} g(x)dx \leq \frac{1}{n}\sum_{i=1}^{k} g\left(\frac{i-1}{n}\right) \leq \frac{1}{n}\, g\left(\frac{k-1}{n}\right) + \int_0^{(k-1)/n} g(x)dx. \quad (6.3.7)$$

Consider the solution x^* where each job $j \in N$ is assigned to $\sigma(j) \in E$. This is in fact the optimal offline solution with value:

$$\begin{aligned}\sum_{i=1}^{n} L_i(x^*)^2 &= \sum_{i=1}^{n} f\left(\frac{i-1}{n}\right)^2 \leq f\left(\frac{n-1}{n}\right)^2 + n\int_0^{(n-1)/n} f(x)^2\, dx \\ &= n + n\int_0^{(n-1)/n} \frac{1}{1-x}dx = n + n\left(\log(1) - \log\left(\frac{1}{n}\right)\right) \\ &= n\,(\log(n) + 1). \end{aligned} \quad (6.3.8)$$

Let us now fix an arbitrary deterministic online fractional algorithm $\mathcal{A}$ generating a solution that we will denote by x. At the arrival of $j \in N$, due to the

random permutation, we have that $\mathbb{E}[x_{\sigma(i),j}] = \mathbb{E}[x_{\sigma(i'),j}]$ for every $i, i' \geq j$. Since the fractional algorithm exactly sends a total fractional value of one, we get that:

$$\mathbb{E}[x_{\sigma(i),j}] = \frac{1}{n-j+1} \qquad \forall i \in \{j, \ldots, n\}.$$

We therefore get that for every $i \in E$:

$$\begin{aligned}\mathbb{E}\Big[L_{\sigma(i)}(x)\Big] &= \sum_{j=1}^{i} w_{\sigma(i),j}\, \mathbb{E}[x_{\sigma(i),j}] = \sum_{j=1}^{i} f\left(\frac{j-1}{n}\right) \frac{1}{n-j+1} \geq \int_0^{(i-1)/n} \frac{f(x)}{1-x} dx \\ &= \int_0^{(i-1)/n} (1-x)^{-3/2}\, dx = \frac{2}{\sqrt{1-x}}\Bigg|_0^{(i-1)/n} = 2\, f\left(\frac{i-1}{n}\right) - 2.\end{aligned}$$

By using Jensen's inequality, we can now lower bound the value obtained by the algorithm:

$$\begin{aligned}\sum_{i=1}^{n} \mathbb{E}\left[L_{\sigma(i)}(x)^2\right] &\geq \sum_{i=1}^{n} \mathbb{E}\left[L_{\sigma(i)}(x)\right]^2 \geq \sum_{i=1}^{n} \left(2\, f\left(\frac{i-1}{n}\right) - 2\right)^2 \\ &\geq 4 \sum_{i=1}^{n} \left(f\left(\frac{i-1}{n}\right)^2 - 2\, f\left(\frac{i-1}{n}\right)\right) \\ &\geq 4\, n \int_0^{(n-1)/n} \Big(f(x)^2 - 2f(x)\Big) dx \\ &= 4n \log(n) - 16n \left(1 - \sqrt{\frac{1}{n}}\right). \qquad (6.3.9)\end{aligned}$$

We thus easily see that the competitive ratio tends to 4 from below when tending n to infinity. For a fixed $\varepsilon > 0$, picking n to be large enough finishes the proof. □

6.3.5 A lower bound for independent rounding algorithms

In this section, we show that any randomized algorithm making *independent* random choices for each job $j \in N$ cannot be better than 5-competitive. This shows that Algorithm 6.3.2 is optimal for this class of randomized algorithms.

Theorem 6.3.11. *For any $\varepsilon > 0$, there exists an instance to the online unrelated load balancing problem such that any randomized algorithm making independent random choices for every job has a competitive ratio of at least $5 - \varepsilon$.*

Proof:
Let us fix such a randomized algorithm $\mathcal{A}$. The instance is exactly the same as in the proof of Theorem 6.3.10. Let us denote by $X_{ij} \in \{0, 1\}$ the indicator

random variable of whether j is assigned to i. Note that we now have two sources of randomness: both the random permutation and the random choices of the algorithm. The cost of a randomized algorithm can be written as follows:

$$\sum_{i\in E}\mathbb{E}\left[L_i(X)^2\right] = \sum_{i\in E}\mathbb{E}\left[L_i(X)\right]^2 + \sum_{i\in E}\mathrm{Var}[L_i(X)]$$

where the expectation is both over the random permutation and the random choices of the algorithm. Now note that, for a fixed permutation σ, we can interpret $x_{\sigma(i),j} := \mathop{\mathbb{E}}_{\mathcal{A}}[X_{\sigma(i),j}]$ as a fractional algorithm, where the expectation is only over the random choices of $\mathcal{A}$. By (6.3.9), we then have that

$$\sum_{i\in E}\mathbb{E}\left[L_i(X)\right]^2 = \sum_{i\in E}\mathbb{E}\left[L_{\sigma(i)}(X)\right]^2 \geq 4n\log(n) - 16n\left(1-\sqrt{\frac{1}{n}}\right). \tag{6.3.10}$$

For the second term, note that

$$\begin{aligned}\sum_{i\in E}\mathrm{Var}[L_i(X)] &= \sum_{i\in E}\sum_{j,k\in N} w_j\, w_k\Big(\mathbb{E}[X_{ij}X_{ik}] - \mathbb{E}[X_{ij}]\mathbb{E}[X_{ik}]\Big)\\ &= \sum_{i\in E}\sum_{j\in N} w_j^2\left(\mathbb{E}[X_{ij}] - \mathbb{E}[X_{ij}]^2\right)\end{aligned}$$

where the last equality uses $\mathbb{E}[X_{ij}X_{ik}] = \mathbb{E}[X_{ij}]\mathbb{E}[X_{ik}]$ for $j \neq k$, due to our independence assumption. Let us first compute a lower bound for the first term:

$$\sum_{i\in E}\sum_{j\in N} w_j^2\,\mathbb{E}[X_{ij}] = \sum_{j\in N} w_j^2 = \sum_{j=1}^{n} f\left(\frac{j-1}{n}\right)^2 \geq n\int_0^{(n-1)/n} f(x)^2dx = n\log(n)$$

where we use $\sum_{i\in E}\mathbb{E}[X_{ij}] = 1$ for the first equality and the approximation (6.3.7) for the inequality. Note that at the arrival of $j \in N$, due to the random permutation, we have:

$$\mathbb{E}[X_{\sigma(i),j}] = \mathop{\mathbb{E}}_{\sigma}\left[x_{\sigma(i),j}\right] = \frac{1}{n-j+1} \qquad \forall i \in \{j,\ldots,n\}.$$

Therefore, we can upper bound the second term as follows:

$$\begin{aligned}\sum_{i\in E}\sum_{j\in N} w_j^2\,\mathbb{E}[X_{ij}]^2 &= \sum_{j\in N} w_j^2\sum_{i\in E}\mathbb{E}[X_{\sigma(i),j}]^2 = n\sum_{j=1}^{n}\sum_{i=j}^{n}\frac{1}{(n-j+1)^3}\\ &= n\sum_{j=1}^{n}\frac{1}{(n-j+1)^2} = n\sum_{j=1}^{n}\frac{1}{j^2} < \frac{\pi^2}{6}n\end{aligned}$$

where the last inequality uses that $\sum_{j=1}^{\infty} 1/j^2 = \pi^2/6$. We thus have that

$$\sum_{i\in E}\mathrm{Var}[L_i(X)] > n\log(n) - \frac{\pi^2}{6}n. \tag{6.3.11}$$

We have seen that the optimal solution has cost $n(\log(n)+1)$ in (6.3.8). Hence, by (6.3.10) and (6.3.11), we see that the competitive ratio tends to 5 as n tends to infinity. □

6.4 Online scheduling under Smith's Rule

6.4.1 The greedy algorithm

We consider and analyze the algorithm GREEDY for the hypergraph model. Whenever a job $j \in N$ arrives, GREEDY picks $i \in \mathcal{S}_j$ (i.e. sets $x_{ij} = 1$) which gives the least increase in the global objective function. The key property of the greedy algorithm is the following lemma. We denote by $C^{(j)}(x)$ the total cost of the algorithm after arrival of job $j \in N$.

Lemma 6.4.1. *For any adversarial instance of the above online scheduling problem, and any solution $(x_{ij})_{j\in N, i\in\mathcal{S}_j}$ obtained by GREEDY, the following inequalities are satisfied for all $j \in N$:*

$$C^{(j)}(x) - C^{(j-1)}(x) \leq \sum_{e\in i}\left(w_j\, p_{ej} + \sum_{k<j} w_j\, w_k\, \min\{\delta_{ej}, \delta_{ek}\}\, z_{ek}\right) \qquad \forall i \in \mathcal{S}_j.$$

Proof:
Consider the online arrival of $j \in N$. At that moment in time, the total cost summed over all resources is $C^{(j-1)}(x)$. Let us analyze the increase in cost if j were to pick any $i \in \mathcal{S}_j$. For every resource $e \in i$, the weighted completion time of j gives a contribution of

$$w_j\Big(p_{ej} + \sum_{k\leq j, k\prec_e j} p_{ek}\, z_{ek}\Big).$$

Moreover, the only jobs for which the completion time is modified on that resource are the already arrived jobs $k \leq j$ assigned to that resource which have a higher Smith ratio, since their completion time is pushed further by p_{ej} due to the entrance of j. Hence, the increase in objective due to those jobs is:

$$\sum_{k\leq j, k\succ_e j} w_k\, p_{ej}\, z_{ek}.$$

Now, observe that by definition of the Smith ratio $\delta_{ek} = p_{ek}/w_k$, the total increase in objective on every resource $e \in i$ (i.e. the sum of the two above quantities) can be written as:

$$w_j\, p_{ej} + \sum_{k<j} w_j\, w_k\, \min\{\delta_{ej}, \delta_{ek}\}\, z_{ek} \qquad \forall e \in i.$$

The total increase in cost then sums this quantity over every resource $e \in i$. By definition, GREEDY will pick $i \in \mathcal{S}_j$ which gives the smallest increase in the objective function, leading to the statement of the Lemma. □

We are now ready to analyze the competitive ratio of GREEDY. We will do so by doing a dual fitting argument on the following semidefinite program, which we used extensively in the previous chapter, and which we called *(SDP-SR)*.

$$\max \sum_{j \in N} y_j - \frac{1}{2}\|v_0\|^2 \tag{6.4.1}$$

$$y_j \leq \sum_{e \in i} w_j \, p_{ej} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij} \rangle \qquad \forall j \in N, \forall i \in \mathcal{S}_j$$

$$\langle v_{ij}, v_{i'k} \rangle \leq \sum_{e \in i \cap i'} w_j \, w_k \, \min\{\delta_{ej}, \delta_{ek}\} \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in N$$

Theorem 6.4.2. *For any instance of the above online scheduling problem, and any solution $(x_{ij})_{j \in N, i \in \mathcal{S}_j}$ obtained by* GREEDY*, there exists a feasible (SDP-SR) solution with objective value at least $C(x)/4$. By weak duality, this implies that the competitive ratio of* GREEDY *is at most* 4.

Proof:
We assume that the SDP vectors live in the inner product space $\mathcal{F}(E)$, which is without loss of generality by Lemma 5.3.1. Let us fix $\beta = 1/2$, we now state the dual fitting for *(SDP-SR)*:

- $v_0(e,t) := -\beta \sum_{k \in N} w_k \, z_{ek} \, \mathbb{1}_{\{t \leq \delta_{ek}\}}$
- $v_{ij}(e,t) := \; w_j \, \mathbb{1}_{\{e \in i\}} \, \mathbb{1}_{\{t \leq \delta_{ej}\}} \qquad \forall j \in N, \forall i \in \mathcal{S}_j$
- $y_j := \beta \left(C^{(j)}(x) - C^{(j-1)}(x) \right) \qquad \forall j \in N.$

Let us now compute the different inner products and norms that we need to argue feasibility of the solution. For a job $j \in N$ and a strategy $i \in \mathcal{S}_j$, we have

$$\|v_{ij}\|^2 = \sum_{e \in i} w_j^2 \, \delta_{ej} = \sum_{e \in i} w_j \, p_{ej}.$$

In addition, for any $(i,j) \neq (i',k)$ with $j,k \in N$, we have

$$\langle v_{ij}, v_{i'k} \rangle = \sum_{e \in E} w_j w_k \mathbb{1}_{\{e \in i\}} \mathbb{1}_{\{e \in i'\}} \int_0^\infty \mathbb{1}_{\{t \leq \delta_{ej}\}} \mathbb{1}_{\{t \leq \delta_{ek}\}} dt = \sum_{e \in i \cap i'} w_j w_k \min\{\delta_{ej}, \delta_{ek}\} \tag{6.4.2}$$

and observe that this tighly satisfies the second SDP constraint. Finally,

$$\langle v_0, v_{ij} \rangle = -\beta \sum_{e \in i} \sum_{k \in N} w_j \, w_k \, z_{ek} \, \min\{\delta_{ej}, \delta_{ek}\}.$$

Let us now check that this is a feasible solution to *(SDP-SR)*. The second set of constraints is satisfied due to (6.4.2). The first set of constraints under the above fitting becomes:

$$\begin{aligned} & y_j \leq \sum_{e \in i} w_j \, p_{ej} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij} \rangle \\ \iff & \beta \left(C^{(j)}(x) - C^{(j-1)}(x) \right) \leq \frac{1}{2} \sum_{e \in i} w_j \, p_{ej} + \beta \sum_{e \in i} \sum_{k \in N} w_j \, w_k \, z_{ek} \, \min\{\delta_{ej}, \delta_{ek}\}. \end{aligned}$$

By the choice $\beta = 1/2$, these inequalities are satisfied by Lemma 6.4.1, implying that the fitted solution is feasible.

Let us now argue about the objective function. Let us denote

$$\eta(x) := \sum_{e \in E} \sum_{j \in N} w_j \, p_{ej} \, z_{ej}.$$

For the norm squared of v_0, we get

$$\begin{aligned} \frac{1}{\beta^2} \, \|v_0\|^2 &= \sum_{e \in E} \sum_{j,k \in N} w_j \, w_k \, z_{ej} \, z_{ek} \int_0^\infty \mathbb{1}_{\{t \leq \delta_{ej}\}} \mathbb{1}_{\{t \leq \delta_{ek}\}} \, dt \\ &= \sum_{e \in E} \sum_{j,k \in N} w_j w_k z_{ej} z_{ek} \min\{\delta_{ej}, \delta_{ek}\} \\ &= \sum_{e \in E} \sum_{j \in N} w_j^2 \, z_{ej}^2 \, \delta_{ej} + 2 \sum_{e \in E} \sum_{j \in N, k \prec_e j} w_j w_k z_{ej} z_{ek} \delta_{ek} \\ &= \sum_{e \in E} \sum_{j \in N} w_j \, p_{ej} \, z_{ej}^2 + 2 \sum_{e \in E} \sum_{j \in N, k \prec_e j} w_j \, p_{ek} \, z_{ej} \, z_{ek} \\ &= 2 \, C(x) - \eta(x). \end{aligned} \tag{6.4.3}$$

The last equality uses the definition of the Smith Ratio $\delta_{ej} = p_{ej}/w_j$, whereas the last inequality follows from the fact that $z_{ej}^2 = z_{ej}$ (since $z_{ej} \in \{0, 1\}$) as well as the definition of the total cost (5.4.2). The sum of the y variables becomes:

$$\sum_{j \in N} y_j = \beta \sum_{j \in N} \left(C^{(j)}(x) - C^{(j-1)}(x) \right) = \beta \, C^{(n)}(x) = \beta \, C(x)$$

where the second equality uses the fact that the sum is telescoping. The objective function of this SDP can now be lower bounded using (6.4.3):

$$\sum_{j \in J} y_j - \frac{1}{2}\|v_0\|^2 = \beta C(x) - \beta^2 C(x) + \beta^2 \, \frac{\eta(x)}{2} = \frac{C(x)}{4} + \frac{\eta(x)}{8} \geq \frac{C(x)}{4}.$$

□

6.4.2 An alternative randomized algorithm

In this section, we provide an alternative 4-competitive randomized algorithm for the online problem $R||\sum w_j C_j$, for which the *(SDP-SR)* relaxation specializes as:

$$\begin{aligned}
\max \sum_{j\in N} y_j - \frac{1}{2}\|v_0\|^2 & \\
y_j \le w_j p_{ij} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij}\rangle & \qquad \forall j \in N, \forall i \in \mathcal{S}_j \\
\langle v_{ij}, v_{i'k}\rangle \le w_j\, w_k\, \min\{\delta_{ij}, \delta_{ik}\}\, \mathbb{1}_{\{i=i'\}} & \qquad \forall (i,j) \neq (i',k) \text{ with } j,k \in N.
\end{aligned}$$

Since we are now considering a randomized algorithm, we will refer to $X_{ij} \in \{0,1\}$ as the binary random variable indicating whether $j \in N$ is assigned to $i \in E$. Moreover, we will refer to $x_{ij} := \mathbb{E}[X_{ij}] \in [0,1]$ as the probabilities defined by the algorithm. The random completion time of a job $j \in N$ is thus

$$C_j(X) = \sum_{i\in E} \sum_{k \preceq_i j} p_{ik}\, X_{ij}\, X_{ik}.$$

The total cost is the sum of weighted completion times:

$$C(X) := \sum_{j\in N} w_j\, C_j(X) = \sum_{i\in E} \sum_{j\in N, k\preceq_i j} w_j\, p_{ik}\, X_{ij}\, X_{ik}. \tag{6.4.4}$$

We can split this cost on a machine by machine basis by defining:

$$L_i(X) = \sum_{j\in N, k\preceq_i j} w_j\, p_{ik}\, X_{ij}\, X_{ik} \qquad \forall i \in E. \tag{6.4.5}$$

Clearly, we have that $C(X) = \sum_{i\in E} L_i(X)$. Since the jobs arrive online, we denote them as $N = \{1,\dots,n\}$, where job k arrives before job j if $k < j$.

Let us now describe the algorithm. Whenever $j \in N$ arrives, we consider the following potential functions for every $i \in \mathcal{S}_j$:

$$f_{ij}(t) := w_j\, p_{ij} + 2\, \left(t\, w_j\, p_{ij} + \sum_{k<j} w_j\, w_k \min\{\delta_{ij}, \delta_{ik}\}\, x_{ik} \right). \tag{6.4.6}$$

We now consider a waterfilling type of algorithm, described in Algorithm 6.4.1, which defines a probability distribution $(x_{ij})_{i\in S_j}$ ensuring that

$$x_{ej} > 0 \implies f_{ej}(x_{ej}) \le f_{ij}(x_{ij}) \qquad \forall i \in \mathcal{S}_j.$$

For convenience of notation, we also let $x_{ej} = 0$ if $e \notin \mathcal{S}_j$. Observe that this means that for every $e \in \mathcal{S}_j$ with $x_{ej} > 0$, we get that $f_{ej}(x_{ej}) = \lambda$ for some

constant λ, whereas $f_{ej}(x_{ej}) \geq \lambda$ if $x_{ej} = 0$. In particular, since $\sum_{e \in \mathcal{S}_j} x_{ej} = 1$, we also get the following equilibrium inequality:

$$\sum_{e \in E} x_{ej}\, f_{ej}(x_{ej}) \leq f_{ij}(x_{ij}) \qquad \forall i \in \mathcal{S}_j. \tag{6.4.7}$$

Once the algorithm has constructed the distribution $(x_{ij})_{i \in \mathcal{S}_j}$, it simply randomly samples a machine from this distribution to assign the job to. Note that the random decisions are independent for any two jobs $j \neq k$.

Algorithm 6.4.1 Waterfilling and randomized rounding

when $j \in N$ **arrives:**

Compute $(x_{ij})_{i \in \mathcal{S}_j}$ such that $\sum_i x_{ij} = 1$ and (6.4.7) holds
Assign j to $i \in \mathcal{S}_j$, i.e. set $X_{ij} = 1$ with probability x_{ij}

return X

By a slight abuse of notation, let us denote the expected cost per resource $i \in E$ as

$$L_i(x) := \mathbb{E}[L_i(X)] = \sum_{j \in N, k \preceq_i j} w_j\, p_{ik}\, \mathbb{E}[X_{ij}\, X_{ik}] \tag{6.4.8}$$

and let us also define the following quantity for every $i \in E$:

$$F_i(x) := \sum_{j \in N, k \preceq_i j} w_j\, p_{ik}\, x_{ij}\, x_{ik}. \tag{6.4.9}$$

Since the algorithm makes independent choices for any two jobs $j \neq k$, meaning that $\mathbb{E}[X_{ij}\, X_{ik}] = \mathbb{E}[X_{ij}]\mathbb{E}[X_{ik}] = x_{ij}x_{ik}$, the only difference between $L_i(x)$ and $F_i(x)$ occurs when $j = k$, in which case we respectively get a contribution of $\mathbb{E}[X_{ij}^2] = \mathbb{E}[X_{ij}] = x_{ij}$ and x_{ij}^2.

Let us now define $L_i^{(j)}(x)$ and $F_i^{(j)}(x)$ to be the expressions (6.4.8) and (6.4.9) for the assignment x after an online job j has arrived (i.e. where $x_{ik} = 0$ for every $k > j$ and every $i \in M$). We now define the increments over time of these two quantities as:

$$\delta_i^{(j)}(x) := F_i^{(j)}(x) - F_i^{(j-1)}(x) \quad \text{and} \quad \Delta_i^{(j)}(x) := L_i^{(j)}(x) - L_i^{(j-1)}(x).$$

We are now ready to state key inequalities satisfied by our algorithm which will be needed for the dual fitting.

Lemma 6.4.3. *For any solution $(x_{ij})_{j \in N, i \in \mathcal{S}_j}$ constructed by Algorithm 6.4.1, the following inequalities are satisfied for every $j \in N$:*

$$\sum_{e \in M} \left(\delta_e^{(j)}(x) + \Delta_e^{(j)}(x)\right) \leq w_j\, p_{ij} + 2 \sum_{k \in N} w_j\, w_k\, \min\{\delta_{ij}, \delta_{ik}\}\, x_{ik} \qquad \forall i \in \mathcal{S}_j.$$

Proof:
Let us first understand how the cost increases when $j \in N$ is assigned to $i \in E$ by the algorithm. The weighted completion time of j increases the cost by:

$$w_j\Big(p_{ij} + \sum_{k<j:k\prec_i j} p_{ik}X_{ik}\Big).$$

Moreover, the completion time of the already arrived jobs assigned to that machine coming after j in the ordering of that machine (i.e. with a higher Smith ratio) is increased by p_{ij}, leading to an increase in the objective of

$$\sum_{k<j:k\succ_i j} w_k\, p_{ij}\, X_{ik}.$$

The total increase in cost (i.e. the sum of the two above quantities) can be written succinctly as follows:

$$w_j p_{ij} + \sum_{k<j} w_j w_k \min\{\delta_{ij}, \delta_{ik}\} X_{ik}.$$

Using that, we can see that the increments become:

$$\begin{aligned}
\Delta_e^{(j)}(x) &= x_{ej}\left(w_j\, p_{ej} + \sum_{k<j} w_j w_k \min\{\delta_{ej}, \delta_{ek}\} x_{ek}\right), \\
\delta_e^{(j)}(x) &= x_{ej}\left(w_j\, p_{ej}\, x_{ej} + \sum_{k<j} w_j w_k \min\{\delta_{ej}, \delta_{ek}\} x_{ek}\right) \\
&= x_{ej}\left(\sum_{k\le j} w_j w_k \min\{\delta_{ej}, \delta_{ek}\} x_{ek}\right)
\end{aligned}$$

where the last equality follows from $w_j p_{ej} = w_j^2 \delta_{ej}$. From the above two equations, we see that $\delta_e^{(j)}(x) + \Delta_e^{(j)}(x) \le x_{ej} f_{ej}(x_{ej})$, by definition of (6.4.6). The equilibrium condition (6.4.7) then tells us that

$$\sum_{e\in E}\big(\delta_e^{(j)}(x) + \Delta_e^{(j)}(x)\big) \le \sum_{e\in E} x_{ej}\, f_{ej}(x_{ej}) \le f_{ij}(x_{ij}) \quad \forall i \in \mathcal{S}_j.$$

Looking again at the definition (6.4.6) completes the proof of the lemma. □

We are now ready to analyze the competitive ratio of the algorithm in this model.

Theorem 6.4.4. *For any online instance and any solution $(x_{ij})_{j\in N, i\in\mathcal{S}_j}$ obtained by Algorithm 6.4.1, there exists a feasible (SDP-SR) solution with objective value at least $\mathbb{E}[C(X)]/4$. By weak duality, this implies that the competitive ratio of the algorithm is at most 4.*

Proof:
We assume that the SDP vectors belong to the space $\mathcal{F}(E)$, which is without loss of generality by Lemma 5.3.1. We now state the dual fitting:

- $v_0(i,t) := -\frac{1}{2} \sum_{k \in N} w_k \, x_{ik} \, \mathbb{1}_{\{t \leq \delta_{ik}\}}$
- $v_{ij}(i',t) := \; w_j \, \mathbb{1}_{\{t \leq \delta_{ij}\}} \mathbb{1}_{\{i=i'\}} \qquad \forall j \in N, \forall i \in \mathcal{S}_j$
- $y_j := \frac{1}{4} \sum_{e \in M} \Big(\delta_e^{(j)}(x) + \Delta_e^{(j)}(x) \Big) \qquad \forall j \in N.$

The desired inner products can be computed to be the following, using essentially the same computations as in Theorem 5.4.1:

$$\langle v_0, v_{ij} \rangle = -\frac{1}{2} \sum_{k \in N} w_j \, w_k \, \min\{\delta_{ij}, \delta_{ik}\} \, x_{ik}, \quad \|v_{ij}\|^2 = w_j \, p_{ij}$$

$$\|v_0\|^2 = \frac{1}{4} \sum_{j,k \in N} w_j \, w_k \, \min\{\delta_{ij}, \delta_{ik}\} \, x_{ij} \, x_{ik}, \quad \langle v_{ij}, v_{i'k} \rangle = \; w_j \, w_k \, \min\{\delta_{ij}, \delta_{ik}\} \, \mathbb{1}_{\{i=i'\}}.$$

The second set of SDP constraints is tightly satisfied due to the last computation above. The first set of constraints under this fitting gives:

$$\begin{aligned} y_j &\leq w_j p_{ij} - \frac{1}{2}\|v_{ij}\|^2 - \langle v_0, v_{ij} \rangle \\ &\iff \frac{1}{4} \sum_{e \in M} \big(\delta_e^{(j)}(x) + \Delta_e^{(j)}(x) \big) \leq \frac{1}{2} \, w_j \, p_{ij} + \frac{1}{2} \sum_{k \in N} w_j \, w_k \, \min\{\delta_{ij}, \delta_{ik}\} \, x_{ik}. \end{aligned}$$

These are now satisfied by Lemma 6.4.3. To argue about the objective, observe that

$$\sum_{j \in N} y_j = \frac{1}{4} \sum_{e \in M} \sum_{j \in N} \big(\delta_e^{(j)}(x) + \Delta_e^{(j)}(x) \big) = \frac{1}{4} \sum_{e \in M} \Big(L_e(x) + F_e(x) \Big)$$

by using the definition of the increments and the fact that the sum is telescoping. In addition,

$$\|v_0\|^2 = \frac{1}{4} \sum_{j,k \in N} w_j \, w_k \, \min\{\delta_{ij}, \delta_{ik}\} \, x_{ij} \, x_{ik} \leq \frac{1}{2} \sum_{e \in M} F_e(x).$$

Hence, the objective can be lower bounded as:

$$\sum_{j \in N} y_j - \frac{1}{2}\|v_0\|^2 \geq \frac{1}{4} \sum_{e \in M} \Big(L_e(x) + F_e(x) \Big) - \frac{1}{4} \sum_{e \in M} F_e(x) = \frac{1}{4} \sum_{e \in M} L_e(x) = \frac{1}{4} \mathbb{E}[C(X)].$$

□

6.4.3 A matching lower bound

In this section, we prove a matching lower bound even against fractional algorithms for this model. To do so, we will use the hard instance constructed for the unrelated load balancing problem in Theorem 6.3.10. We now describe the intuition of how the two models are related in the restricted identical machines setting, meaning that $w_{ij} \in \{w_j, \infty\}$. Given an instance and a fractional solution $x \in [0,1]^{M \times N}$, the square of the L_2 norm of the loads is:

$$C^{(LB)}(x) := \sum_{i \in M} \left(\sum_{j \in N} w_j x_{ij} \right)^2 .$$

Consider now the same instance in the second model under uniform Smith ratios, meaning that the processing times are set to $p_{ij} = w_j$ for every $i \in \mathcal{S}_j$. Given a fractional solution x, the cost in this model can be computed to be:

$$C^{(SR)}(x) := \frac{1}{2} C^{(LB)}(x) + \sum_{i \in M} \sum_{j \in N} w_j^2 \left(x_{ij} - \frac{1}{2} x_{ij}^2 \right). \tag{6.4.10}$$

The idea of the proof will now be to modify the instance in Theorem 6.3.10 in order to make the second term above negligible.

Consider the instance $\mathcal{I}$ constructed in Theorem 6.3.10. We now consider a modified instance $\mathcal{I}(t)$, which for each job $j \in N$, has $t \in \mathbb{N}$ copies of the same job arriving consecutively, all of which have the same feasible machines. These copies of a job j are denoted as $K(j)$. Moreover, for every $k \in K(j)$, we set the weight to be $w_k = w_j/t$. The new set of jobs is denoted by $\tilde{N} = \cup_{j \in N} K(j)$.

Lemma 6.4.5. *For a fractional solution $y \in [0,1]^{M \times \tilde{N}}$ to $\mathcal{I}(t)$ and a fractional solution $x \in [0,1]^{M \times N}$ to $\mathcal{I}$ satisfying $\sum_{k \in K(j)} y_{ik} = t\, x_{ij}$ for every $j \in N$ and every $i \in M$, we have*

$$C^{(SR)}(y) = \frac{1}{2} C^{(LB)}(x) + O\left(\frac{1}{t}\right).$$

Proof:

By (6.4.10), we have that

$$\begin{aligned}
C^{(SR)}(y) &= \frac{1}{2}\sum_{i\in M}\left(\sum_{j\in N}\sum_{k\in K(j)} w_k y_{ik}\right)^2 + \sum_{i\in M}\sum_{j\in N}\sum_{k\in K(j)} w_k^2\left(y_{ik} - \frac{1}{2}y_{ik}^2\right) \\
&= \frac{1}{2}\sum_{i\in M}\left(\sum_{j\in N}\frac{w_j}{t}\sum_{k\in K(j)} y_{ik}\right)^2 + \sum_{i\in M}\sum_{j\in N}\frac{w_j^2}{t^2}\sum_{k\in K(j)}\left(y_{ik} - \frac{1}{2}y_{ik}^2\right) \\
&= \frac{1}{2}\sum_{i\in M}\left(\sum_{j\in N} w_j x_{ij}\right)^2 + \frac{1}{t}\sum_{j\in N} w_j^2 - \frac{1}{t^2}\sum_{i\in M}\sum_{j\in N} w_j^2 \sum_{k\in K(j)} y_{ik}^2 \\
&= \frac{1}{2}\,C^{(LB)}(x) + O\left(\frac{1}{t}\right).
\end{aligned}$$

□

Theorem 6.4.6. *For any $\varepsilon > 0$, there exists an online instance to the online scheduling problem under Smith's Rule such that any fractional algorithm has a competitive ratio of at least $4 - \varepsilon$.*

Proof:
Note that we can easily convert a solution x of $\mathcal{I}$ into a solution of $\mathcal{I}(t)$ by setting $y_{ik} = x_{ij}$ for every job $k \in K(j)$. By Lemma 6.4.5, this shows that the optimal solution y^* of instance $\mathcal{I}(t)$ has cost at most

$$C^{(SR)}(y^*) \le \frac{1}{2}\,C^{(LB)}(x^*) + O\left(\frac{1}{t}\right) = \frac{1}{2}\,n\,(\log(n)+1) + O\left(\frac{1}{t}\right)$$

where x^* is the optimal solution on instance $\mathcal{I}$ whose cost we computed in (6.3.8). Conversely, we can convert a fractional solution y obtained by an arbitrary online algorithm on $\mathcal{I}(t)$ to a solution generated online on $\mathcal{I}$ by setting $x_{ij} = \sum_{k\in K(j)} y_{ik}/t$. By (6.3.9), the cost of that solution on $\mathcal{I}$ is at least:

$$C^{(LB)}(x) \ge 4n\log(n) - 16n\left(1 - \sqrt{\frac{1}{n}}\right).$$

By Lemma 6.4.5, we have that any online fractional algorithm incurs a cost of at least

$$C^{(SR)}(y) \ge \frac{1}{2}\left(4n\log(n) - 16n\left(1 - \sqrt{\frac{1}{n}}\right)\right) + O\left(\frac{1}{t}\right).$$

Hence, by letting n and t tend to infinity, we get the proof of the theorem. □

Chapter 7

A faster algorithm for explorable heap selection

In this chapter, we study the explorable heap selection problem. The goal is to find the nth smallest value in a binary heap, and the complexity of the algorithm is measured by the total distance traveled in the tree, with each edge having unit cost. This problem was introduced by Karp, Saks and Widgerson [KSW86], who gave deterministic and randomized $n\exp(O(\sqrt{\log n}))$ time algorithms using $O(\log(n)^{2.5})$ and $O(\sqrt{\log n})$ space respectively. We present a new randomized algorithm with running time $O(n\log(n)^3)$ against an oblivious adversary using $O(\log n)$ space, substantially improving the previous best randomized running time at the expense of slightly increased space usage.

7.1 Introduction

Many important problems in theoretical computer science are fundamentally search problems. The objective of these problems is to find a certain solution from the search space. In this chapter, we analyze a search problem that we call *explorable heap selection*. The problem is related to the famous branch-and-bound algorithm and was originally proposed by Karp, Saks and Widgerson [KSW86] to model node selection for branch-and-bound with low space-complexity. Furthermore, as we will explain later, the problem remains practically relevant to branch-and-bound even in the full space setting.

The explorable heap selection problem[1] is an online graph exploration problem for an agent on a rooted (possibly infinite) binary tree. The nodes of the tree are labeled by distinct real numbers (the key values) that increase along every path starting from the root. The tree can thus be thought of as a min-heap. Starting at the root, the agent's objective is to select the n^{th} smallest value in the tree while minimizing the distance traveled, where each edge of the tree has

[1] [KSW86] did not give the problem a name, so we have attempted to give a descriptive one here.

unit travel cost. The key value of a node is only revealed when the agent visits it, and thus the problem has an online nature. When the agent learns the key value of a node, it still does not know the rank of this value.

The selection problem for ordinary heaps, which allow for random access (i.e., jumping to arbitrary nodes in the tree for "free"), has also been studied. In this model, it was shown by [Fre93] that selecting the n^{th} minimum can be achieved deterministically in $O(n)$ time using $O(n)$ workspace. We note that in both models, $\Omega(n)$ is a natural lower bound. This is because verifying that a value $\mathcal{L}$ is the n^{th} minimum requires $\Theta(n)$ time – one must at least inspect the n nodes with value at most $\mathcal{L}$ – which can be done via straightforward depth-first search.

A simple selection strategy is to use the best-first rule[2], which repeatedly explores the unexplored node whose parent has the smallest key value. While this rule is optimal in terms of the number of nodes that it explores, namely $\Theta(n)$, the distance traveled by the agent can be far from optimal. In the worst-case, an agent using this rule will need to travel a distance of $\Theta(n^2)$ to find the n^{th} smallest value. A simple bad example for this rule is to consider a rooted tree consisting of two paths (which one can extend to a binary tree), where the two paths are consecutively labeled by all positive even and odd integers respectively. Moreover, the space complexity becomes $\Omega(n)$ in general when using the best-first rule, because essentially all the explored nodes might need to be kept in memory. We note that irrespective of computational considerations on the agent, either in terms of working memory or running time restrictions, minimizing the total travel distance in explorable heap selection remains a challenging online problem.

Improving on the best-first strategy, Karp, Saks and Wigderson [KSW86] gave a randomized algorithm with expected cost $n \cdot \exp(O(\sqrt{\log(n)}))$ using $O(\sqrt{\log(n)})$ working space. They also showed how to make the algorithm deterministic using $O(\log(n)^{2.5})$ space. In this work, our main contribution is an improved randomized algorithm with expected cost $O(n \log(n)^3)$ using $O(\log(n))$ space. Given the $\Omega(n)$ lower bound, our travel cost is optimal up to logarithmic factors. Furthermore we show that any algorithm for explorable heap selection that uses only s units of memory, must take at least $n \cdot \log_s(n)$ time in expectation. An interesting open problem is the question whether a superlinear lower bound also holds without any restriction on the memory usage.

To clarify the memory model, it is assumed that any key value and $O(\log n)$ bit integer can be stored using $O(1)$ space. We also assume that maintaining the current position in the tree does not take up memory. Furthermore, we assume that key value comparisons and moving across an edge of the tree require $O(1)$ time. Under these assumptions, the running times of the above algorithms happen to be proportional to their travel cost. Throughout this chapter, we will thus use travel cost and running time interchangeably.

[2]Frederickson's algorithm [Fre93] is in fact a highly optimized variant of this rule

Motivation. The motivation to look at this problem comes from the branch-and-bound algorithm. This is a well-known algorithm that can be used for solving many types of problems. In particular, it is often used to solve integer linear programs (IPs), which are of the form $\arg\min\{c^\top x : x \in \mathbb{Z}^n, Ax \leq b\}$. In that setting, branch-and-bound works by first solving the linear programming (LP) relaxation, which does not have integrality constraints. The value of the solution to the relaxation forms a lower bound on the objective value of the original problem. Moreover, if this solution only has integral components, it is also optimal for the original problem. Otherwise, the algorithm chooses a component x_i for which the solution value $\hat{x}_i$ is not integral. It then creates two new subproblems, by either adding the constraint $x_i \leq \lfloor \hat{x}_i \rfloor$ or $x_i \geq \lceil \hat{x}_i \rceil$. This operation is called *branching*. The tree of subproblems, in which the children of a problem are created by the branching operation, is called the branch-and-bound tree. Because a subproblem contains more constraints than its parent, its objective value is greater or equal to the one of its parent. The algorithm can also be used to solve mixed-integer linear programs (MIPs), where some of the variables are allowed to be continuous.

At the core, the algorithm consists of two important components: the branching rule and the node selection rule. The branching rule determines how to split up a problem into subproblems, by choosing a variable to branch on. Substantial research has been done on branching rules, see, e.g., [LS99, AKM05, LZ17, BDSV18].

The node selection rule decides which subproblem to solve next. Not much theoretical research has been done on the choice of the node selection rule. Traditionally, the best-first strategy is thought to be optimal from a theoretical perspective because this rule minimizes the number of nodes that need to be visited. However, a disadvantage of this rule is that searches using it might use space proportional to the number of explored nodes, because all of them need to be kept in memory. In contrast to this, a simple strategy like depth-first search only needs to store the current solution. Unfortunately, performing a depth-first search can lead to an arbitrarily bad running time. This was the original motivation for introducing the explorable heap selection problem [KSW86]. By guessing the number N of branch-and-bound nodes whose LP values are at most that of the optimal IP solution (which can be done via successive doubling), a search strategy for this problem can be directly interpreted as a node selection rule. The algorithm that they introduced can therefore be used to implement branch-and-bound efficiently in only $O\left(\sqrt{\log(N)}\right)$ space.

Nowadays, computers have a lot of memory available. This usually makes it feasible to store all explored nodes of the branch-and-bound tree in memory. However, many MIP-solvers still make use of a hybrid method that consists of both depth-first and best-first searches. This is not only done because depth-first search uses less memory, but also because it is often faster. Experimental studies

have confirmed that the depth-first strategy is in many cases faster than best-first one [CP99]. This seems contradictory, because the running time of best-first search is often thought to be theoretically optimal.

In part, this contradiction can be explained by the fact that actual IP-solvers often employ complementary techniques and heuristics on top of branch-and-bound, which might benefit from depth-first searches. Additionally, a best-first search can hop between different parts of the tree, while a depth first search subsequently explores nodes that are very close to each other. In the latter case, the LP-solver can start from a very similar state, which is known as warm starting. This is faster for a variety of technical reasons [Ach09]. For example, this can be the case when the LP-solver makes use of the LU-factorization of the optimal basis matrix [MJSS16]. Through the use of dynamic algorithms, computing this can be done faster if a factorization for a similar LP-basis is known [SS93]. Because of its large size, MIP-solvers will often not store the LU-factorization for all nodes in the tree. This makes it beneficial to move between similar nodes in the branch-and-bound tree. Furthermore, moving from one part of the tree to another means that the solver needs to undo and redo many bound changes, which also takes up time. Hence, the amount of distance traveled between nodes in the tree is a metric that influences the running time. This can also be observed when running the academic MIP-solver SCIP [Gle22].

The explorable heap selection problem captures these benefits of locality by measuring the running time in terms of the amount of travel through the tree. Therefore, we argue that this problem is still relevant for the choice of a node selection rule, even if all nodes can be stored in memory.

Related work. The explorable heap selection problem was first introduced in [KSW86]. Their result was later applied to prove an upper bound on the parallel running time of branch-and-bound [PPSV15].

When random access to the heap is provided at constant cost, selecting the n'th value in the heap can be done by a deterministic algorithm in $O(n)$ time by using an additional $O(n)$ memory for auxilliary data structures [Fre93].

The explorable heap selection problem can be thought of as a *search game* [AG03] and bears some similarity to the *cow path problem*. In the cow path problem, an agent explores an unweighted unlabeled graph in search of a target node. The location of the target node is unknown, but when the agent visits a node they are told whether or not that node is the target. The performance of an algorithm is judged by the ratio of the number of visited nodes to the distance of the target from the agent's starting point. In both the cow path problem and the explorable heap selection problem, the cost of backtracking and retracing paths is an important consideration. The cow path problem on infinite b-ary trees was studied in [DCD95] under the assumption that when present at a node the agent can obtain an estimate on that node's distance to the target.

Other explorable graph problems exist without a target, where typically the graph itself is unknown at the outset. There is an extensive literature on exploration both in graphs and in the plane [Ber98, Tho06]. In some of the used models the objective is to minimize the distance traveled [BCGL23, BDHS23, MMS12, KP94]. Other models are about minimizing the amount of used memory [DFKP04]. What distinguises the explorable heap selection problem from these problems is the information that the graph is a heap and that the ordinal of the target is known. This can allow an algorithm to rule out certain locations for the target. Because of this additional information, the techniques used here do not seem to be applicable to these other problems.

Outline of the chapter. In Section 7.2 we formally introduce the explorable heap selection problem and any notation we will use. In Section 7.3 we introduce a new algorithm for solving this problem and provide a running time analysis.

7.2 The explorable heap selection problem

We introduce in this section the formal model for the explorable heap selection problem. The input to the algorithm is an infinite binary tree $T = (V, E)$, where each node $v \in V$ has an associated real value, denoted by $\mathrm{val}(v) \in \mathbb{R}$. We assume that all the values are distinct. Moreover, for each node in the tree, the values of its children are larger than its own value. Hence, for every $v_1, v_2 \in V$ such that v_1 is an ancestor of v_2, we have that $\mathrm{val}(v_2) > \mathrm{val}(v_1)$. The binary tree T is thus a heap.

The algorithmic problem we are interested in is finding the n^{th} smallest value in this tree. This may be seen as an online graph exploration problem where an agent can move in the tree and learns the value of a node each time they explore it. At each time step, the agent resides at a vertex $v \in V$ and may decide to move to either the left child, the right child or the parent of v (if it exists, i.e. if v is not the root of the tree). Each traversal of an edge costs one unit of time, and the complexity of an algorithm for this problem is thus measured by the total traveled distance in the binary tree. The algorithm is also allowed to store values in memory.

We now introduce a few notations used throughout this chapter.

- For a node $v \in V$, also per abuse of notation written $v \in T$, we denote by $T^{(v)}$ the subtree of T rooted at v.
- For a tree T and a value $\mathcal{L} \in \mathbb{R}$, we define the subtree $T_{\mathcal{L}} := \{v \in T \mid \mathrm{val}(v) \leq \mathcal{L}\}$.
- We denote the n^{th} smallest value in T by $\mathsf{SELECT}^T(n)$. This is the quantity that we are interested in finding algorithmically.

- We say that a value $\mathcal{V} \in \mathbb{R}$ is *good* for a tree T if $\mathcal{V} \leq \mathsf{SELECT}^T(n)$ and *bad* otherwise. Similarly, we call a node $v \in T$ *good* if $\text{val}(v) \leq \mathsf{SELECT}^T(n)$ and *bad* otherwise.
- We will use $[k]$ to refer to the set $\{1, \ldots, k\}$.
- When we write $\log(n)$, we assume the base of the logarithm to be 2.

For a given value $\mathcal{V} \in \mathbb{R}$, it is easy to check whether it is good in $O(n)$ time: start a depth first search at the root of the tree, turning back each time a value strictly greater than $\mathcal{V}$ is encountered. In the meantime, count the number of values below $\mathcal{V}$ found so far and stop the search if more than n values are found. If the number of values below $\mathcal{V}$ found at the end of the procedure is at most n, then $\mathcal{V}$ is a good value. This procedure is described in more detail later in the DFS subroutine.

We will often instruct the agent to move to an already discovered good vertex $v \in V$. The way this is done algorithmically is by saving $\text{val}(v)$ in memory and starting a depth first search at the root, turning back every time a value strictly bigger than $\text{val}(v)$ is encountered until finally finding $\text{val}(v)$. This takes at most $O(n)$ time, since we assume v to be a good node. If we instruct the agent to go back to the root from a certain vertex $v \in V$, this is simply done by traveling back in the tree, choosing to go to the parent of the current node at each step.

In later sections, we will often say that a subroutine takes a subtree $T^{(v)}$ as input. This implicitly means that we in fact pass it $\text{val}(v)$ as input, make the agent travel to $v \in T$ using the previously described procedure, call the subroutine from that position in the tree, and travel back to the original position at the end of the execution. Because the subroutine knows the value $\text{val}(v)$ of the root of $T^{(v)}$, it can ensure it never leaves the subtree $T^{(v)}$, thus making it possible to recurse on a subtree as if it were a rooted tree by itself. We write the subtree $T^{(v)}$ as part of the input for simplicity of presentation.

We will sometimes want to pick a value uniformly at random from a set of values $\{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$ of unknown size that arrives in a streaming fashion, for instance when we traverse a part of the tree T by doing a depth first search. That is, we see the value $\mathcal{V}_i$ at the i^{th} time step, but do not longer have access to it in memory once we move on to $\mathcal{V}_{i+1}$. This can be done by generating random values $\{X_1, \ldots, X_k\}$ where, at the i^{th} time step, $X_i = \mathcal{V}_i$ with probability $1/i$, and $X_i = X_{i-1}$ otherwise. It is easy to check that X_k is a uniformly distributed sample from $\{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$.

7.3 A new algorithm

The authors of [KSW86] presented a deterministic algorithm that solves the explorable heap selection problem in $n \cdot \exp(O(\sqrt{\log(n)}))$ time and $O(n\sqrt{\log(n)})$

space. By replacing the binary search that is used in the algorithm by a randomized variant, they are able to decrease the space requirements. This way, they obtain a randomized algorithm with expected running time $n \cdot \exp(O(\sqrt{\log(n)}))$ and space complexity $O(\sqrt{\log(n)})$. Alternatively, the binary search can be implemented using a deterministic routine by [MP80] to achieve the same running time with $O(\log(n)^{2.5})$ space.

We present a randomized algorithm with a running time $O(n \log(n)^3)$ and space complexity $O(\log(n))$. Unlike the algorithms mentioned before, our algorithm fundamentally relies on randomness to bound its running time. This bound only holds when the algorithm is run on a tree with labels that are fixed before the execution of the algorithm. That is, the tree must be generated by an adversary that is oblivious to the choices made by the algorithm. This is a stronger assumption than is needed for the algorithm that is given in [KSW86], which also works against adaptive adversaries. An adaptive adversary is able to defer the decision of the node label to the time that the node is explored. Note that this distinction does not really matter for the application of the algorithm as a node selection rule in branch-and-bound, since there the node labels are fixed because they are derived from the integer program and branching rule.

Theorem 7.3.1. *There exists a randomized algorithm that solves the explorable heap selection problem, with expected running time $O(n \log(n)^3)$ and $O(\log(n))$ space.*

As mentioned above, checking whether a value v is good can be done in $O(n)$ time by doing a depth-first search with cutoff value $\text{val}(v)$ that returns when more than n good nodes are found. For a set of k values, we can determine which of them are good in $O(\log(k)n)$ time by performing a binary search.

The explorable heap selection problem can be seen as the problem of finding all n good nodes. Both our method and that of [KSW86] function by first identifying a subtree consisting of only good nodes. The children of the leaves of this subtree are called "roots" and the subtree is extended by finding a number of new good nodes under these roots in multiple rounds. Importantly, the term 'good node' is always used with respect to the current call to EXTEND. So, a node might be good in one recursive call, but not good in another.

In [KSW86] this is done by running $O(c^{\sqrt{2\log(n)}})$ different rounds, for some constant $c > 1$. In each round, the algorithm finds $n/c^{\sqrt{2\log(n)}}$ new good nodes. These nodes are found by recursively exploring each active root and using binary search on the observed values to discover which of these values are good. Which active roots are recursively explored further depends on which values are good. The recursion in the algorithm is at most $O(\sqrt{\log(n)})$ levels deep, which is where the space complexity bound comes from.

In our algorithm, we take a different approach. We will call our algorithm consecutively with $n = 1, 2, 4, 8, \ldots$. Hence, for a call to the algorithm, we can

assume that we have already found at least $n/2$ good nodes. These nodes form a subtree of the original tree T. In each round, our algorithm chooses a random root under this subtree and finds every good node under it. It does so by doing recursive subcalls to the main algorithm on this root with values $n = 1, 2, 4, 8, \ldots$. As soon as the recursively obtained node is a bad node, the algorithm stops searching the subtree of this root, since it is guaranteed that all the good nodes there have been found. The largest good value that is found can then be used to find additional good nodes under the other roots without recursive calls, through a simple depth-first search. Assuming that the node values were fixed in advance, we expect this largest good value to be greater than half of the other roots' largest good values. Similarly, we expect its smallest bad value to be smaller than half of the other roots' smallest bad values. By this principle, a sizeable fraction of the roots can, in expectation, be ruled out from getting a recursive call. Each round a new random root is selected until all good nodes have been found.

This algorithm allows us to effectively perform binary search on the list of roots, ordered by the largest good value contained in each of their subtrees in $O(\log n)$ rounds, and the same list ordered by the smallest bad values (Lemma 7.3.5). Bounding the expected number of good nodes found using recursive subcalls requires a subtle induction on two parameters (Lemma 7.3.4): both n and the number of good nodes that have been identified so far.

7.3.1 Subroutines

We first describe three subroutines that will be used in our main algorithm.

The procedure DFS. The procedure DFS is a variant of depth first search. The input to the procedure is T, a cutoff value $\mathcal{L} \in \mathbb{R}$ and an integer $n \in \mathbb{N}$. The procedure returns the number of vertices in T whose value is at most $\mathcal{L}$.

It achieves that by exploring the tree T in a depth first search manner, starting at the root and turning back as soon as a node $w \in T$ such that $\mathrm{val}(w) > \mathcal{L}$ is encountered. Moreover, if the number of nodes whose value is at most $\mathcal{L}$ exceeds n during the search, the algorithm stops and returns $n + 1$.

The algorithm output is the following integer.

$$\mathrm{DFS}(T, \mathcal{L}, n) := \min\left\{\left|T_{\mathcal{L}}\right|, n + 1\right\}.$$

Observe that the DFS procedure allows us to check whether a node $w \in T$ is a good node, i.e. whether $\mathrm{val}(w) \leq \mathsf{SELECT}^T(n)$. Indeed, w is good if and only if $\mathrm{DFS}(T, \mathrm{val}(w), n) \leq n$.

This algorithm visits only nodes in $T_{\mathcal{L}}$ or its direct descendants and its running time is $O(n)$. The space complexity is $O(1)$, since the only values needed to be stored in memory are $\mathcal{L}$, $\mathrm{val}(v)$, where v is the root of the tree T, and a counter for the number of good values found so far.

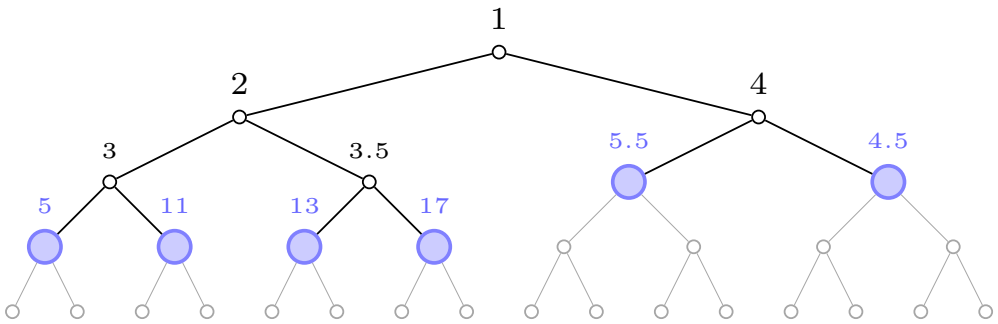


Figure 7.1: An illustration of $R(T, \mathcal{L}_0)$ with $\mathcal{L}_0 = 4$. The number above each vertex is its value, the blue nodes are $R(T, \mathcal{L}_0)$, whereas the subtree above is $T_{\mathcal{L}_0}$.

The procedure Roots. The procedure Roots takes as input a tree T as well as an initial fixed lower bound $\mathcal{L}_0 \in \mathbb{R}$ on the value of $\mathsf{SELECT}^T(n)$. We assume that the main algorithm has already found all the nodes $w \in T$ satisfying $\mathrm{val}(w) \leq \mathcal{L}_0$. This means that the remaining values the main algorithm needs to find in T are all lying in the subtrees of the following nodes, that we call the *$\mathcal{L}_0$-roots of T*:

$$R(T, \mathcal{L}_0) := \left\{ r \in T \setminus T_{\mathcal{L}_0} \;\middle|\; r \text{ is a child of a node in } T_{\mathcal{L}_0} \right\}$$

In other words, these are all the vertices in T one level deeper in the tree than $T_{\mathcal{L}_0}$, see Figure 7.1 for an illustration. In addition to that, the procedure takes two other parameters $\mathcal{L}, \mathcal{U} \in \mathbb{R}$ as input, which correspond to (another) lower and upper bound on the value of $\mathsf{SELECT}^T(n)$. These bounds $\mathcal{L}$ and $\mathcal{U}$ will be variables being updated during the execution of the main algorithm, where $\mathcal{L}$ will be increasing and $\mathcal{U}$ will be decreasing. More precisely, $\mathcal{L}$ will be the largest value that the main algorithm has certified being at most $\mathsf{SELECT}^T(n)$, whereas $\mathcal{U}$ will be the smallest value that the algorithm has certified being at least that. A key observation is that these lower and upper bounds can allow us to remove certain roots in $R(T, \mathcal{L}_0)$ from consideration, in the sense that all the good values in that root's subtree will be certified to have already been found. The only roots that the main algorithm needs to consider, when $\mathcal{L}$ and $\mathcal{U}$ are given, are thus the following.

$$\textsc{Roots}(T, \mathcal{L}_0, \mathcal{L}, \mathcal{U}) := \left\{ r \in R(T, \mathcal{L}_0) \mid \exists w \in T^{(r)} \text{ with } \mathrm{val}(w) \in (\mathcal{L}, \mathcal{U}) \right\} \quad (7.3.1)$$

This subroutine can be implemented as follows. Run a depth-first search starting at the root of T. Once a node $r \in T$ with $\mathrm{val}(r) > \mathcal{L}_0$ is encountered, the subroutine marks that vertex r as belonging to $R(T, \mathcal{L}_0)$. The depth-first search continues deeper in the tree until finding a node $w \in T^{(r)}$ with $\mathrm{val}(w) > \mathcal{L}$. At this point, if $\mathrm{val}(w) < \mathcal{U}$, then the search directly returns to r without exploring any additional nodes in $T^{(r)}$ and adds r to the output. If however $\mathrm{val}(w) \geq \mathcal{U}$,

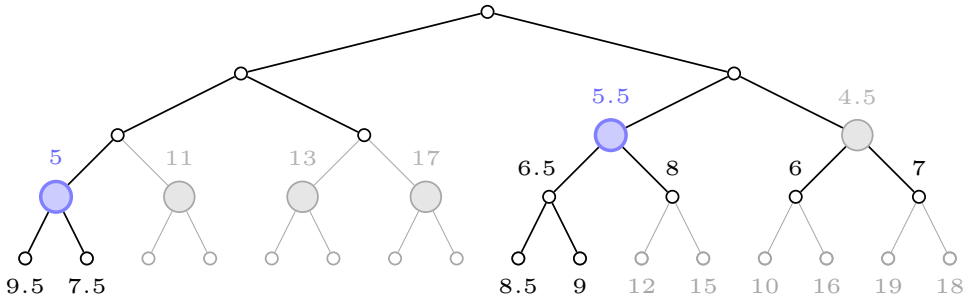


Figure 7.2: An illustration of the ROOTS procedure with $\mathcal{L}_0 = 4, \mathcal{L} = 7$ and $\mathcal{U} = 10$. Only two active roots remain, and are both colored in blue. The other roots are considered killed since all the good values have been found in their subtrees.

then the search continues exploring $T_{\mathcal{L}}^{(r)}$ (and its direct descendants) by trying to find a node w with $\text{val}(w) \in (\mathcal{L}, \mathcal{U})$. In case the algorithm explores all of $T_{\mathcal{L}}^{(r)}$ with its direct descendants, and it turns out that no such node exists (i.e. every direct descendant w of $T_{\mathcal{L}}^{(r)}$ satisfies $\text{val}(w) \geq \mathcal{U}$), then r is not added to the output.

This procedure takes time $O(|T_{\mathcal{L}}|)$, i.e. proportional to the number of nodes in T with value at most $\mathcal{L}$. Since the procedure is called only on values $\mathcal{L}$ which are known to be good, the time is bounded by $O(|T_{\mathcal{L}}|) = O(n)$.

In the main algorithm, we will only need this procedure in order to select a root from $\text{ROOTS}(T, \mathcal{L}_0, \mathcal{L}, \mathcal{U})$ uniformly at random, without having to store the whole list in memory. This can then be achieved in $O(1)$ space, since one then only needs to store $\text{val}(v), \mathcal{L}_0, \mathcal{L}$ and $\mathcal{U}$ in memory, where v is the root of the tree T.

The procedure GoodValues. The procedure GOODVALUES takes as input a tree T, a subtree $T^{(r)}$ for a node $r \in T$, a value $\mathcal{L}' \in \mathbb{R}_{\geq 0}$ and an integer $n \in \mathbb{N}$. The procedure then analyzes the set

$$S := \left\{ \text{val}(w) \;\middle|\; w \in T^{(r)}, \text{val}(w) \leq \mathcal{L}' \right\}$$

and outputs both the largest good value and the smallest bad value in that set, that we respectively call $\mathcal{L}$ and $\mathcal{U}$. If no bad values exist in S, the algorithm sets $\mathcal{U} = \infty$. Notice that this output determines, for each value in S, whether it is good or not. Indeed, any $\mathcal{V} \in S$ is good if and only if $\mathcal{V} \leq \mathcal{L}$, and is bad if and only if $\mathcal{V} \geq \mathcal{U}$.

The implementation is as follows. Start by initializing the variables $\mathcal{L} = -\infty$ and $\mathcal{U} = \mathcal{L}'$. These variables correspond to lower and upper bounds on $\mathsf{SELECT}^T(n)$. Loop through the values in

$$S' := \left\{ \text{val}(w) \mid w \in T^{(r)},\ \mathcal{L} < \text{val}(w) < \mathcal{U} \right\}$$

using a depth first search starting at r and sample one value $\mathcal{V}$ uniformly randomly from that set. Check whether $\mathcal{V}$ is a good value by calling $\mathrm{DFS}(T, \mathcal{V}, n)$. If it is good, update $\mathcal{L} = \mathcal{V}$. If it is bad, update $\mathcal{U} = \mathcal{V}$. Continue this procedure until S' is empty, i.e. $|S'| = 0$. If, at the end of the procedure, $\mathcal{L} = \mathcal{L}' = \mathcal{U}$, then set $\mathcal{U} = \infty$. The output is thus:

$$\textsc{GoodValues}(T, T^{(r)}, \mathcal{L}', n) := \{\mathcal{L}, \mathcal{U}\}$$

where

$$\mathcal{L} := \max\left\{\mathcal{V} \in S \mid \mathcal{V} \leq \mathsf{SELECT}^T(n)\right\},$$
$$\mathcal{U} := \min\left\{\mathcal{V} \in S \mid \mathcal{V} > \mathsf{SELECT}^T(n)\right\}.$$

Sampling a value from S' takes $O(|S|)$ time. Checking whether a sampled valued is good takes $O(n)$ time. In expectation, the number of updates before the set S' is empty is $O(\log(|S|))$, leading to an expected total running time of $O((|S| + n)\log(|S|))$. As we will later see in the proof of Lemma 7.3.6, we will only end up making calls $\textsc{GoodValues}(T, T^{(r)}, \mathcal{L}', n)$ with parameters $T^{(r)}$ and $\mathcal{L}'$ satisfying $\mathrm{DFS}(T^{(r)}, \mathcal{L}') = O(n)$. Since $|S| = \mathrm{DFS}(T^{(r)}, \mathcal{L}')$, this leads to an expected running time of $O(n \log(n))$.

The procedure can be implemented in $O(1)$ space, since the only values needed to be kept in memory are $\mathrm{val}(v)$ (where v is the root of the tree T), $\mathrm{val}(r)$, $\mathcal{L}$, $\mathcal{U}$ and $\mathcal{L}'$, as well as the fact that every call to DFS also requires $O(1)$ space.

7.3.2 The main algorithm

We now present our main algorithm. This algorithm is named Select and outputs the n^{th} smallest value in the tree T. A procedure used in Select is the Extend algorithm, described below, which assumes that at least $n/2$ good nodes have already been found in the tree, and also outputs the n^{th} smallest one.

Let us describe a few invariants from the Extend procedure.

- $\mathcal{L}$ and $\mathcal{U}$ are respectively lower and upper bounds on $\mathsf{SELECT}^T(n)$ during the whole execution of the procedure. More precisely, $\mathcal{L} \leq \mathsf{SELECT}^T(n)$ and $\mathcal{U} > \mathsf{SELECT}^T(n)$ at any point, and hence $\mathcal{L}$ is good and $\mathcal{U}$ is bad. The integer k counts the number of values $\leq \mathcal{L}$ in the full tree T.

- No root can be randomly selected twice. This is ruled out by the updated values of $\mathcal{L}$ and $\mathcal{U}$, and the proof can be found in Theorem 7.3.2.

- After an iteration of the inner while loop, $\mathcal{L}'$ is set to the c^{th} smallest value in $T^{(r)}$. The variable c' then corresponds to the next value we would like to find in $T^{(r)}$ if we were to continue the search. Note that $c' \leq 2c$, enforcing that the recursive call to Extend satisfies its precondition, and

Algorithm 7.3.1 The SELECT procedure

Input : $n \in \mathbb{N}$
Output : SELECT(n), the n^{th} smallest value in the heap T.
procedure SELECT(n)
    $k \leftarrow 1$
    $\mathcal{L} \leftarrow \text{val}(v)$ ▷ v is the root of the tree T
    while $k < n$ **do**
        if $k < n/2$ **then**
            $k' \leftarrow 2k$
        else
            $k' \leftarrow n$
        $\mathcal{L} \leftarrow$ EXTEND$(T, k', k, \mathcal{L})$
        $k \leftarrow k'$
    return $\mathcal{L}$

Algorithm 7.3.2 The EXTEND procedure

Input: T: tree which is to be explored.
        $n \in \mathbb{N}$: total number of good values to be found, satisfying $n \geq 2$.
        $k \in \mathbb{N}$: number of good values already found, satisfying $k \geq n/2$.
        $\mathcal{L}_0 \in \mathbb{R}$: value satisfying $\text{DFS}(T, \mathcal{L}_0, n) = k$.
Output: the n^{th} smallest value in T.
procedure EXTEND$(T,\ n,\ k,\ \mathcal{L}_0)$
    $\mathcal{L} \leftarrow \mathcal{L}_0$
    $\mathcal{U} \leftarrow \infty$
    while $k < n$ **do**
        $r \leftarrow$ random element from ROOTS$(T,\ \mathcal{L}_0,\ \mathcal{L},\ \mathcal{U})$
        $\mathcal{L}' \leftarrow \max(\mathcal{L}, \text{val}(r))$
        $k' \leftarrow \text{DFS}(T,\ \mathcal{L}',\ n)$ ▷ count the number of values $\leq \mathcal{L}'$ in T
        $c \leftarrow \text{DFS}(T^{(r)},\ \mathcal{L}',\ n)$ ▷ counting the number of values $\leq \mathcal{L}'$ in $T^{(r)}$
        $c' \leftarrow \min(n - k' + c, 2c)$ ▷ increase the number of values to be found
        while $k' < n$ **do**
            $\mathcal{L}' \leftarrow$ EXTEND$(T^{(r)},\ c',\ c,\ \mathcal{L}')$
            $k' \leftarrow \text{DFS}(T,\ \mathcal{L}',\ n)$
            $c \leftarrow c'$
            $c' \leftarrow \min(n - k' + c, 2c)$
        $\tilde{\mathcal{L}}, \tilde{\mathcal{U}} \leftarrow$ GOODVALUES$(T, T^{(r)}, \mathcal{L}', n)$ ▷ find the good values in $T^{(r)}$
        $\mathcal{L} \leftarrow \max(\mathcal{L}, \tilde{\mathcal{L}})$
        $\mathcal{U} \leftarrow \min(\mathcal{U}, \tilde{\mathcal{U}})$
        $k \leftarrow \text{DFS}(T,\ \mathcal{L},\ n)$ ▷ compute the number of good values found in T
    return $\mathcal{L}$

that $c' \leq n - (k' - c)$ implies that $(k' - c) + c' \leq n$, which implies that the recursive subcall will not spend time searching for a value that is known in advance to be bad.

- From the definition of k' and c one can see that $k' \geq k + c$. Combined with the previous invariant, we see that $c' \leq n - k$.

- k' always counts the number of values $\leq \mathcal{L}'$ in the full tree T. It is important to observe that this is a global parameter, and does not only count values below the current root. Moreover, $k' \geq n$ implies that we can stop searching below the current root, since it is guaranteed that all good values in $T^{(r)}$ have been found, i.e., $\mathcal{L}'$ is larger than all the good values in $T^{(r)}$.

7.3.3 Proof of correctness

Theorem 7.3.2. *At the end of the execution of Algorithm 7.3.1, $\mathcal{L}$ is set to the n^{th} smallest value in T. Moreover, the algorithm is guaranteed to terminate.*

Proof:
We show $\mathcal{L} = \mathsf{SELECT}^T(n)$ holds at the end of Algorithm 7.3.2, i.e. the EXTEND procedure. Correctness of Algorithm 7.3.1, i.e. the SELECT procedure, then clearly follows from that. First, notice that $\mathcal{L}$ is always set to the first output of the procedure GOODVALUES, which is always the value of a good node in T, implying

$$\mathcal{L} \leq \mathsf{SELECT}^T(n)$$

at any point during the execution of the algorithm. Since the outer while loop ends when at least n good nodes in T have value at most $\mathcal{L}$, we get

$$\mathcal{L} \geq \mathsf{SELECT}^T(n),$$

which implies that when the algorithm terminates it does so with the correct value.

It remains to prove that the algorithm terminates. We observe that every recursive call $\mathcal{L}' \leftarrow \text{EXTEND}(T^{(r)}, c', c, \mathcal{L}')$ strictly increases the value of $\mathcal{L}'$, ensuring that at least one extra value in T is under the increased value. This implies that k' strictly increases every iteration of the inner while loop, thus ensuring that this loop terminates.

To see that the outer loop terminates, observe that after each iteration the set $\text{ROOTS}(T, \mathcal{L}_0, \mathcal{L}, \mathcal{U})$ shrinks by at least one element. As soon as this set is empty, there will be no more roots with unexplored good values in their subtrees, so $k = n$ and the algorithm terminates. □

7.3.4 Space complexity analysis

We prove in this section the space complexity of our main algorithm.

Theorem 7.3.3. *The procedure* Select(n) *runs in* $O(\log(n))$ *space.*

Proof:
Observe that it is enough to prove that the statement holds for Extend$(T, n, k, \mathcal{L})$ with $k \geq n/2$, since the memory can be freed up (only keeping the returned value in memory) after every call to Extend in the Select(n) algorithm.

Moreover, observe that the subroutines DFS, Roots and GoodValues all require $O(1)$ memory, as argued in their respective analyses. Any call to Extend$(T, n, k, \mathcal{L})$ only makes recursive calls to Extend$(T^{(r)}, \hat{n}, \hat{k}, \hat{\mathcal{L}})$ with $1 \leq \hat{n} \leq n - k \leq \frac{1}{2}n$. So the depth of the recursion is at most $\log(n)$, and the space complexity of the algorithm is $O(\log(n))$. □

7.3.5 Running time analysis

In order to prove a $O(n \log(n)^3)$ running time bound for the Select(n) procedure, we will show that the running time of the Extend procedure with parameters n and k is $O((n-k)\log(n)^3) + O(n\log(n)^2)$.

The main challenge in analyzing the running time of Extend is in dealing with the cost of the recursive subcalls on line 16. For this we rely on an important idea, formalized in Lemma 7.3.4, stating that if the parent call with parameters n and k makes $z \in \mathbb{N}$ recursive calls with parameters $(n_1, k_1), \ldots, (n_z, k_z)$, then $\sum_{i=1}^{z}(n_i - k_i) \leq n - k$ in expectation over the random choices of the algorithm.

A second insight is that the outermost while loop on line 9 is executed at most $O(\log(n))$ times in expectation, which is shown in Lemma 7.3.5. The first lemma allows to show that the running time of the Extend procedure on the recursive part is $O((n-k)\log(n)^3)$, through an induction proof. The second lemma helps to show that the running time of the Extend procedure on the non-recursive part is $O(n\log(n)^2)$. The running time analysis of Extend is formally done in Lemma 7.3.6. Finally, the running time of $O(n\log(n)^3)$ for the Select(n) procedure then follows in Theorem 7.3.7.

Let us now prove these claims. We first show that the expectation of $\sum_{i=1}^{z}(n_i - k_i)$ is bounded by $n - k$.

Lemma 7.3.4. *Let z be the number of recursive calls with $k \geq 1$ that are done in the main loop of* Extend*$(T, n, k, \mathcal{L}_0)$. For every $i \in [z]$, let n_i and k_i be the values that are given as second and third parameters to the ith such subcall. It holds that:*

$$\mathbb{E}\left[\sum_{i=1}^{z}(n_i - k_i)\right] \leq n - k.$$

Proof:
For simplicity of notation, let us denote the set of roots at the beginning of the execution of the algorithm by $\mathcal{R} := \textsc{Roots}(T, \mathcal{L}_0, \mathcal{L}, \mathcal{U})$, where $\mathcal{L} = \mathcal{L}_0$ and $\mathcal{U} = \infty$ at initialization. An important observation is that, once a root $r \in \mathcal{R}$ is randomly selected on line 10, all the recursive calls under it (i.e. with its subtree $T^{(r)}$ as first parameter) on line 16 are consecutive. The last such recursive call ensures that all the good values in $T^{(r)}$ are found and sets $\mathcal{L}$ and $\mathcal{U}$ to respectively be the largest good value and smallest bad value in $T^{(r)}$. From then on, this root leaves the updated set $\textsc{Roots}(T, \mathcal{L}_0, \mathcal{L}, \mathcal{U})$ by (7.3.1) and will thus never be again considered in the random choice on line 10. For every $r \in \mathcal{R}$, let us define the set:

$$C(r) = \Big\{ i \in [z] \text{ s.t. the } i\text{th recursive call is under root } r \Big\}$$

and let us denote by $S_r \in \mathbb{N}$ the total number of good values in its subtree $T^{(r)}$. Our goal is to show that:

$$\mathbb{E}\Big[\sum_{i \in C(r)} (n_i - k_i) \Big] \leq S_r \qquad \forall r \in \mathcal{R}. \tag{7.3.2}$$

Clearly, this would imply the lemma, since the total number of good values to be found is $\sum_{r \in \mathcal{R}} S_r = n - k$. For convenience, we define this number to be $p := n - k$. We now order the good values to be found and denote them as follows: $\mathcal{V}_1 < \mathcal{V}_2 < \cdots < \mathcal{V}_p$. Each value $\mathcal{V}_k$ is to be found in the subtree of a certain root that we denote by $r(\mathcal{V}_k) \in \mathcal{R}$.

We first show that the claim (7.3.2) holds for any root $r \in \mathcal{R}$ such that $r \neq r(\mathcal{V}_p)$. Let us thus fix such a root $r \neq r(\mathcal{V}_p)$. The key observation is that, since the random choice on line 10 is uniform, and since $r(\mathcal{V}_p)$ will always be among the active roots, the subtree of the root $r(\mathcal{V}_p)$ will be explored before the subtree of root r with probability at least a half. In that case, no recursive calls will be made under root r. This holds since the updated values $\mathcal{L}$ and $\mathcal{U}$ after the iteration of $r(\mathcal{V}_p)$ ensure that r leaves $\textsc{Roots}(T, \mathcal{L}_0, \mathcal{L}, \mathcal{U})$ by (7.3.1) and is thus not considered in the random choice in later iterations. If the root r is however considered before $r(\mathcal{V}_p)$, which happens with probability at most a half, then $\sum_{i \in C(r)} (n_i - k_i) \leq 2S_r$, since the sum is telescoping and the parameters k_i and n_i at most double at each step on line 19 until all good values in $T^{(r)}$ are found. Hence, we get that

$$\mathbb{E}\Big[\sum_{i \in C(r)} (n_i - k_i) \Big] \leq \frac{1}{2}\, 0 + \frac{1}{2}\, 2S_r \leq S_r. \tag{7.3.3}$$

It remains to show that claim (7.3.2) holds for the root $r(\mathcal{V}_p)$ under which the largest good value lies. In that case, let us denote by $\mathcal{V}_j$ the largest good

value lying in a subtree of a different root $r(\mathcal{V}_j) \neq r(\mathcal{V}_p)$. We also denote by $\{r(\mathcal{V}_j) \prec r(\mathcal{V}_p)\}$ the probabilistic event that $r(\mathcal{V}_j)$ is considered before $r(\mathcal{V}_p)$ in the random choices of the algorithm. By our choice of $\mathcal{V}_j$ and $\mathcal{V}_p$, this event happens with probability exactly a half. Moreover, if this event happens, all the good values outside of $T^{(r(\mathcal{V}_p))}$ will have been found after exploring $T^{(r(\mathcal{V}_j))}$. This means that, when the algorithm considers $r(\mathcal{V}_p)$, it knows that there remain at most $p-j$ values to be found. That is, we will have $C(r(\mathcal{V}_p)) = \{t, \dots, z\}$ for some t, such that $k_t \geq S_{r(\mathcal{V}_p)} - (p-j)$ and $n_z \leq S_{r(\mathcal{V}_p)}$, leading to

$$\mathbb{E}\Big[\sum_{i \in C(r(\mathcal{V}_p))} (n_i - k_i) \;\Big|\; r(\mathcal{V}_j) \prec r(\mathcal{V}_p)\Big] \leq S_{r(\mathcal{V}_p)} - \big(S_{r(\mathcal{V}_p)} - (p-j)\big) = p - j, \tag{7.3.4}$$

where we have again used the fact that the sum is telescoping.

We now consider the event $\{r(\mathcal{V}_p) \prec r(\mathcal{V}_j)\}$ and distinguish two cases. Suppose that the penultimate call $i \in C(r(\mathcal{V}_p))$ finds a good value which is bigger than $\mathcal{V}_j$. By a similar argument as above, the algorithm does not double in the last step, but truncates due to line 19, meaning that $\sum_{i \in C(r(\mathcal{V}_p))}(n_i - k_i) = S_{r(\mathcal{V}_p)}$ holds in this case. Combining this with (7.3.4) and using the fact that the last $p-j$ values are under root $r(\mathcal{V}_p)$, we get

$$\mathbb{E}\Big[\sum_{i \in C(r(\mathcal{V}_p))} (n_i - k_i)\Big] \leq \frac{1}{2}(p-j) + \frac{1}{2}S_{r(\mathcal{V}_p)} \leq S_{r(\mathcal{V}_p)}.$$

Suppose now that the penultimate call $i \in C(r(\mathcal{V}_p))$ finds a good value which is smaller than $\mathcal{V}_j$. This means that the number of good values found in $T^{(r(\mathcal{V}_p))}$ is at most $S_{r(\mathcal{V}_p)} - (p-j)$ at that point. The last call $i \in C(r(\mathcal{V}_p))$ then doubles the parameters, meaning that $\sum_{i \in C(r(\mathcal{V}_p))}(n_i - k_i) \leq 2\left(S_{r(\mathcal{V}_p)} - (p-j)\right)$ holds, due to the fact that the sum is telescoping. Combining this with (7.3.4) leads to

$$\mathbb{E}\Big[\sum_{i \in C(r(\mathcal{V}_p))} (n_i - k_i)\Big] \leq \frac{1}{2}(p-j) + S_{r(\mathcal{V}_p)} - (p-j) \leq S_{r(\mathcal{V}_p)}.$$

□

We now bound the expected number of iterations of the outermost while-loop.

Lemma 7.3.5. *The expected number of times that the outermost while-loop (at line 9) is executed by the procedure* Extend *is at most* $O(\log(n))$.

Proof:
Let $r_1, \dots, r_m$ denote the roots returned by $\textsc{Roots}(T, \mathcal{L}_0, \mathcal{L}_0, \infty)$. For $j \in [m]$, let ℓ_j and u_j respectively denote the largest good value and the smallest non-good value under root r_j. Let $A_\ell(\mathcal{L}) := \{r_j : \ell_j > \mathcal{L}\}$ and $A_u(\mathcal{U}) := \{r_j : u_j < \mathcal{U}\}$. Observe that $\textsc{Roots}(T, \mathcal{L}_0, \mathcal{L}, \mathcal{U}) = A_\ell(\mathcal{L}) \cup A_u(\mathcal{U})$ for any $\mathcal{L} \leq \mathcal{U}$.

Let $\mathcal{L}_i$ and $\mathcal{U}_i$ denote the values of $\mathcal{L}$ and $\mathcal{U}$ at the start of the ith iteration. After an iteration i in which root r_j was selected, the algorithm updates $\mathcal{L}$ and $\mathcal{U}$ such that $\mathcal{L}_{i+1} = \max(\mathcal{L}, \ell_j)$ and $\mathcal{U}_{i+1} = \min(\mathcal{U}, u_j)$. Observe that $\mathcal{L}_i$ is nondecreasing and that $\mathcal{U}_i$ is nonincreasing.

We will now show that if a root from $A_\ell(\mathcal{L}_i)$ is selected in iteration i, then the expected size of $A_\ell(\mathcal{L}_{i+1})$ is at most half that of $A_\ell(\mathcal{L}_i)$. This will imply that in expectation only $\log(n)$ iterations are needed to make $|A_\ell(\mathcal{L})| = 1$.

Let $\mathcal{F}_i$ be the filtration containing all information up until iteration i. Let X_i be a random variable denoting the value of $|A_\ell(\mathcal{L}_i)|$. Let $(s_k)_{k\geq 1}$ be the subsequence consisting of iteration indices i in which a root from $A_\ell(\mathcal{L}_i)$ is selected. Because roots are selected uniformly at random, we have $\mathbb{E}[X_{s_{k+1}} \mid \mathcal{F}_{s_k}] \leq \frac{1}{2} X_{s_k}$.

Let $Y_i = \max(\log(X_i), 0)$. Note that when $Y_{s_k} \geq 1$, we have $\mathbb{E}[Y_{s_{k+1}} \mid \mathcal{F}_{s_k}] = \mathbb{E}[\log(X_{s_{k+1}}) \mid \mathcal{F}_{s_k}] \leq \log(\mathbb{E}[X_{s_{k+1}} \mid \mathcal{F}_{s_k}]) \leq Y_{s_k} - 1$. Let τ be the smallest k such that $Y_{s_k} = 0$. Note that τ is the number of iterations i in which a root from $A_\ell(\mathcal{L}_i)$ is selected, and hence $\tau \leq n$. The sequence $(Y_{s_k} + k)_{k=1,\dots,\tau}$ is therefore a supermartingale and τ is a stopping time. By the martingale stopping theorem [MU05, Theorem 12.2], we have $\mathbb{E}[\tau] = \mathbb{E}[Y_{s_\tau} + \tau] \leq \mathbb{E}[Y_{s_1} + 1] = \log(m) + 1$.

Now we have shown that in expectation at most $\log(m) + 1$ iterations i are needed in which roots from $A_\ell(\mathcal{L}_i)$ are considered. The same argument can be repeated for $A_u(\mathcal{U})$. Since in every iteration a root from $A_\ell(\mathcal{L})$ or $A_u(\mathcal{U})$ is selected, the expected total number of iterations is at most $2\log(m) + 2$. This directly implies the lemma as $m \leq |T_\mathcal{L}| + 1 \leq n + 1$. □

We are now able to prove the running time bound for the EXTEND procedure.

Lemma 7.3.6. *Let $R(T, n, k)$ denote the running time of a call EXTEND(T, n, k, $\mathcal{L}_0$). Then there exists $C > 0$ such that*

$$\mathbb{E}[R(T, n, k)] \leq 5C(n - k)\log(n)^3 + Cn\log(n)^2.$$

Proof:
We will prove this with induction on $r := \lceil \log(n) \rceil$. For $r = 1$, we have $n \leq 2$. In this case R is constant, proving our induction base.

Now consider a call EXTEND(T, n, k, $\mathcal{L}_0$) and assume the induction claim is true when $\lceil \log(n) \rceil \leq r - 1$. Let p be the number of iterations of the outer-most while-loop that are executed.

We will first consider the running time induced by the base call itself, excluding any recursive subcalls. Note that all of this running time is incurred by the calls to the procedures DFS, ROOTS and GOODVALUES, plus the cost of moving to the corresponding node before each of these calls. In the base call, the procedure will only move between nodes that are among the ones with the n smallest values, or the nodes directly below them. For this reason, we can upper bound the cost of each movement action by a running time of $O(n)$.

- On line 12, 13, 23 each call DFS incurs a running time of at most $O(n)$.

Each of these lines will be executed p times, incurring a total running time of $O(pn)$.

- On line 17 each call DFS(T, $\mathcal{L}'$, n) incurs a running time of at most $O(n)$. The code will be executed $O(p \log(n))$ times since c' doubles after every iteration of the inner loop and never grows larger than n, thus incurring a total running time of $O(pn \log(n))$.
- The arguments $T^{(r)}$ and $\mathcal{L}'$ of the call to GoodValues on line 20 satisfy $\text{DFS}(T^{(r)}, \mathcal{L}') = c \leq c' \leq n$. Hence, the running time of this procedure is $O(n \log(n))$ time. The line is executed at most p times, so the total running time incurred is $O(pn \log(n))$.

Adding up all the running times listed before, we see that the total running time incurred by the non-recursive part is $O(pn \log(n))$. By Lemma 7.3.5, $\mathbb{E}[p] \leq \log(n)$. Hence, we can choose C such that the expected running time of the non-recursive part is bounded by

$$Cn \log(n)^2.$$

Now we move on to the recursive part of the algorithm. All calls to Extend(T, n, k, $\mathcal{L}_0$) with $k = 0$ will have $n = 1$, so each of these calls takes only $O(1)$ time. Hence we can safely ignore these calls.

Let z be the number of of recursive calls to Extend(T, n, k, $\mathcal{L}_0$) that are done from the base call with $k \geq 1$. Let T_i, k_i, n_i for $i \in [z]$ be the arguments of these function calls. Note that $n/2 \geq n - k \geq n_i \geq 2$ for all i. By the induction hypothesis, the expectation of the recursive part of the running time is:

$$\begin{aligned}
&\mathbb{E}\left[\sum_{i=1}^{z} R(T_i, n_i, k_i)\right] \leq \mathbb{E}\left[\sum_{i=1}^{z} 5C(n_i - k_i)\log(n_i)^3 + Cn_i \log(n_i)^2\right] \\
&\leq 5C \log(n/2)^3 \mathbb{E}\left[\sum_{i=1}^{z}(n_i - k_i)\right] + C \log(n/2)^2 \mathbb{E}\left[\sum_{i=1}^{z} n_i\right] \\
&\leq 5C(\log(n) - 1)\log(n)^2 \mathbb{E}\left[\sum_{i=1}^{z}(n_i - k_i)\right] + C \log(n)^2 \mathbb{E}\left[\sum_{i=1}^{z} n_i\right] \\
&\leq 5C(\log(n) - 1)\log(n)^2(n - k) + 5C \log(n)^2(n - k) \\
&\leq 5C(n - k)\log(n)^3.
\end{aligned}$$

Here we used Lemma 7.3.4 as well as the fact that $\sum_{i=1}^{z} n_i \leq 4(n - k)$. To see why the latter inequality is true, consider an arbitrary root r that has S_r values under it that are good (with respect to the base call). We then get

$$\sum_{i=1}^{z} \mathbf{1}_{\{T_i = T^{(r)}\}} n_i \leq \sum_{i=2}^{\lceil \log(S_r+1) \rceil} 2^i \leq 2^{\lceil \log(S_r+1) \rceil + 1} \leq 4S_r.$$

In total there are $n-k$ good values under the roots, and hence $\sum_{i=1}^{z} n_i \leq 4(n-k)$. Adding the expected running time of the recursive and the non-recursive part, we see that

$$\mathbb{E}[R(T,n,k)] \leq 5C(n-k)\log(n)^3 + Cn\log(n)^2.$$

□

This now implies the desired running time for the procedure SELECT.

Theorem 7.3.7. *The procedure* SELECT(n) *runs in expected* $O(n\log(n)^3)$ *time.*

Proof:
The key idea is that SELECT calls EXTEND$(T,k',k,\mathcal{L})$ at most $\lceil\log(n)\rceil$ times with parameters $(k',k) = (2^i, 2^{i-1})$ for $i \in \{1,\ldots,\lceil\log(n)\rceil\}$. By Lemma 7.3.6, the running time of SELECT can thus be upper bounded by

$$\begin{aligned}\sum_{i=1}^{\lceil\log(n)\rceil} \mathbb{E}[R(T,2^i,2^{i-1})] &\leq 5C\log(n)^3 \sum_{i=1}^{\lceil\log(n)\rceil}(2^i - 2^{i-1}) + \sum_{i=1}^{\lceil\log(n)\rceil} Cn\log(n)^2 \\ &= O(n\log(n)^3).\end{aligned}$$

□

Conclusion

In this thesis, we studied three classes of combinatorial optimization problems – covering, matching and scheduling – in different offline, online and game-theoretic models. In addition, we also considered an online graph exploration problem on a heap.

We first looked at the classical vertex cover problem and analyzed a beyond the worst-case approximation algorithm based on the standard linear programming relaxation. The approximation ratio obtained is a bound interpolating between the worst-case bound of 2 and the optimal bound of 1 achievable for bipartite graphs. A key parameter turned out to be the odd girth, which is defined as the length of the shortest odd cycle and which we used as a measure of how far the input graph was from being bipartite. As a byproduct, the techniques developed showed how to get tight bounds on the integrality gap of the standard LP relaxation for three-colorable graphs. An interesting question would be to apply similar techniques to other combinatorial optimization problems in different beyond worst-case models. We believe such models offer a different viewpoint and can give new insights, even for well studied classical problems.

We then looked at a generalization of the online bipartite matching problem to hypergraphs. We focused on the three dimensional version under vertex arrivals and presented a primal-dual fractional algorithm which is $(e-1)/(e+1) \approx 0.462$-competitive. As our main contribution, we then showed that this algorithm is optimal by constructing an adversarial instance adaptive to the actions of an arbitrary fractional algorithm establishing a matching upper bound. This instance combines ideas from two hard instances for online matching on bipartite graphs under the edge arrival and vertex arrival models. An outstanding open question is still to improve the bound of $1/3$ for the integral setting, which is achieved by the greedy algorithm. We believe this setting is considerably harder than for bipartite graphs, and advanced dependent/correlated rounding techniques will be required to make progress on this problem. One could also look at different arrival models, or k-uniform hypergraphs with $k > 3$. When treating k as a large parameter, tight bounds on the competitive ratio are however known up to small constant factors, since the fractional version admits upper and lower bounds of $\Theta(1/\log k)$, whereas the integral version does so with bounds of $\Theta(1/k)$.

We then developed a dual fitting framework based on a single semidefinite pro-

gram allowing to tightly analyze the price of anarchy of games, the approximation ratio of local search algorithms and the competitive ratio of online algorithms for scheduling problems under the sum of weighted completion times objective. The dual fitting approach roughly consisted of making a certain set of SDP constraints correspond to respectively Nash equilibrium inequalities, local optima inequalities, or inequalities satisfied by an online algorithm at every time step. This framework allowed us to derive simple and unified proofs of numerous results in all three of these settings.

Interesting open questions which arise from this work are for instance the following. What is the best coordination ratio achievable by a coordination mechanism for the scheduling problem $R||\sum w_j C_j$? The current best bound is $32/15 \approx 2.133$ achieved by the *Rand* policy. How good can combinatorial algorithms do on this problem? The current best approximation ratio is $(5+\sqrt{5})/4+\varepsilon \approx 1.809+\varepsilon$, which is still quite far from the bound of $1.36+\varepsilon$ achievable by rounding a convex program. It would be interesting to apply this technique to other assignment problems whose optimal solution can be modeled as binary quadratic program. Can this perhaps be extended to problems with a higher degree polynomial objective by considering later rounds of the SDP hierarchy? We believe these are all possible directions for future research.

To conclude, we have in this thesis explored techniques to prove tight bounds on the approximability of different combinatorial optimization problems, often using linear programming or semidefinite programming duality. We believe that there are still many interesting open questions to study in this rich research area, which may lead to the development of new interesting algorithmic techniques. We hope that some of the results in this thesis might have a small contribution in that direction.

Bibliography

[AAE05] Baruch Awerbuch, Yossi Azar, and Amir Epstein. The price of routing unsplittable flow. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 57–66, 2005.

[AAG+95] Baruch Awerbuch, Yossi Azar, Edward F Grove, Ming-Yang Kao, P Krishnan, and Jeffrey Scott Vitter. Load balancing in the Lp norm. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 383–391. IEEE, 1995.

[ABD+23] Itai Ashlagi, Maximilien Burq, Chinmoy Dutta, Patrick Jaillet, Amin Saberi, and Chris Sholley. Edge-weighted online windowed matching. *Math. Oper. Res.*, 48(2):999–1016, 2023.

[ABL02] Sanjeev Arora, Béla Bollobás, and László Lovász. Proving integrality gaps without knowing the linear program. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 313–322. IEEE, 2002.

[ACE+20] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *International Conference on Machine Learning*, pages 345–355. PMLR, 2020.

[Ach09] Tobias Achterberg. *Constraint Integer Programming.* PhD thesis, TU Berlin, 2009.

[ACH14] Fidaa Abed, José R Correa, and Chien-Chung Huang. Optimal coordination mechanisms for multi-job scheduling games. In *Algorithms-ESA 2014: 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings 21*, pages 13–24. Springer, 2014.

[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 *Symposium on the Theory of Computing*, 2005.

[ADG+11] Sebastian Aland, Dominic Dumrauf, Martin Gairing, Burkhard Monien, and Florian Schoppmann. Exact price of anarchy for polynomial congestion games. *SIAM Journal on Computing*, 40(5):1211–1233, 2011.

[AE05] Yossi Azar and Amir Epstein. Convex programming for scheduling unrelated parallel machines. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 331–337, 2005.

[AG03] Steve Alpern and Shmuel Gal. *The Theory of Search Games and Rendezvous*. Number 55 in International Series in Operations Research & Management Science. Kluwer Academic Publishers, Boston, 2003.

[AGK12] S Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1228–1241. SIAM, 2012.

[AGKK20] Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. *Advances in Neural Information Processing Systems*, 33:7933–7944, 2020.

[AGKM11] Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1253–1264. SIAM, 2011.

[AH12] Fidaa Abed and Chien-Chung Huang. Preemptive coordination mechanisms for unrelated machines. In *European Symposium on Algorithms*, pages 12–23. Springer, 2012.

[AJM08] Yossi Azar, Kamal Jain, and Vahab Mirrokni. (almost) optimal coordination mechanisms for unrelated machine scheduling. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 323–332, 2008.

[AKM05] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, January 2005.

[AWZ17] Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1206–1219, 2017.

[BCGL23] Siddhartha Banerjee, Vincent Cohen-Addad, Anupam Gupta, and Zhouzi Li. Graph searching with predictions. In *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, Saarbrücken/Wadern, 2023. Schloss-Dagstuhl.

[BCJS74] James Bruno, Edward G Coffman Jr, and Ravi Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7):382–387, 1974.

[BDHK23] Sander Borst, Daniel Dadush, Sophie Huiberts, and Danish Kashaev. A nearly optimal randomized algorithm for explorable heap selection. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 29–43. Springer, 2023.

[BDHK24] Sander Borst, Daniel Dadush, Sophie Huiberts, and Danish Kashaev. A nearly optimal randomized algorithm for explorable heap selection. *Mathematical Programming*, pages 1–22, 2024.

[BDHS23] Júlia Baligács, Yann Disser, Irene Heinrich, and Pascal Schweitzer. Exploration of graphs with excluded minors. In *31st Annual European Symposium on Algorithms (ESA 2023)*, Saarbrücken/Wadern, 2023. Schloss Dagstuhl.

[BDSV18] Maria-Florina Balcan, Travis Dick, T. Sandholm, and Ellen Vitercik. Learning to branch. *ICML*, 2018.

[Ber98] Piotr Berman. *On-line searching and navigation*, pages 232–241. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[BFMRV14] Adrian Bock, Yuri Faenza, Carsten Moldenhauer, and Andres Jacinto Ruiz-Vargas. Solving the stable set problem in terms of the odd cycle packing number. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[BFPS19] Abbas Bazzi, Samuel Fiorini, Sebastian Pokutta, and Ola Svensson. No small linear program approximates vertex cover within a factor 2- ε. *Mathematics of Operations Research*, 44(1):147–172, 2019.

[BFR98] R Balasubramanian, Michael R Fellows, and Venkatesh Raman. An improved fixed-parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.

[BG93] Jonathan F Buss and Judy Goldsmith. Nondeterminism within p. *SIAM Journal on Computing*, 22(3):560–572, 1993.

[BGR14] Kshipra Bhawalkar, Martin Gairing, and Tim Roughgarden. Weighted congestion games: the price of anarchy, universal worst-case examples, and tightness. *ACM Transactions on Economics and Computation (TEAC)*, 2(4):1–23, 2014.

[BIKM14] Sayan Bhattacharya, Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Coordination mechanisms from (almost) all scheduling policies. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 121–134, 2014.

[BJN07] Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *European Symposium on Algorithms*, pages 253–264. Springer, 2007.

[BK25] Sander Borst and Danish Kashaev. Improved online load balancing in the two-norm. *arXiv preprint arXiv:2511.03345*, 2025.

[BKK25] Sander Borst, Danish Kashaev, and Zhuan Khye Koh. Online matching on 3-uniform hypergraphs. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 100–113. Springer, 2025.

[BKM14] Sayan Bhattacharya, Janardhan Kulkarni, and Vahab Mirrokni. Coordination mechanisms for selfish routing over time on a tree. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I 41*, pages 186–197. Springer, 2014.

[BM08] Benjamin Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *Acm Sigact News*, 39(1):80–87, 2008.

[BMS20] Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. *Advances in Neural Information Processing Systems*, 33:20083–20094, 2020.

[BN09] Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Mathematics of Operations Research*, 34(2):270–286, 2009.

[BP03] Nikhil Bansal and Kirk Pruhs. Server scheduling in the lp norm: a rising tide lifts all boat. In *Proceedings of the thirty-fifth annual acm symposium on theory of computing*, pages 242–250, 2003.

[BSS16] Nikhil Bansal, Aravind Srinivasan, and Ola Svensson. Lift-and-round to improve weighted completion time on unrelated machines. In *Proceedings of the forty-eighth annual acm symposium on theory of computing*, pages 156–167, 2016.

[BST19] Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online algorithms for maximum cardinality matching with edge arrivals. *Algorithmica*, 81(5):1781–1799, 2019.

[BYE81] Reuven Bar-Yehuda and Shimon Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981.

[BYE83] Reuven Bar-Yehuda and Shimon Even. A local-ratio theorm for approximating the weighted vertex cover problem. Technical report, Computer Science Department, Technion, 1983.

[Car08] Ioannis Caragiannis. Better bounds for online load balancing on unrelated machines. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 972–981. Citeseer, 2008.

[Car13] Ioannis Caragiannis. Efficient coordination mechanisms for unrelated machine scheduling. *Algorithmica*, 66(3):512–540, 2013.

[CCG+11] Richard Cole, José R Correa, Vasilis Gkatzelis, Vahab Mirrokni, and Neil Olver. Inner product spaces for minsum coordination mechanisms. In *Proceedings of the forty-third annual ACM symposium on theory of computing*, pages 539–548, 2011.

[CDNK11] Johanne Cohen, Christoph Dürr, and Thang Nguyen Kim. Non-clairvoyant scheduling games. *Theory of Computing Systems*, 49:3–23, 2011.

[CF19] Ioannis Caragiannis and Angelo Fanelli. An almost ideal coordination mechanism for unrelated machine scheduling. *Theory of Computing Systems*, 63:114–127, 2019.

[CFH+20] Michele Conforti, Samuel Fiorini, Tony Huynh, Gwenaël Joret, and Stefan Weltge. The stable set problem in graphs with bounded genus and bounded odd cycle packing number. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2896–2915. SIAM, 2020.

[CFK+06] Ioannis Caragiannis, Michele Flammini, Christos Kaklamanis, Panagiotis Kanellopoulos, and Luca Moscardelli. Tight bounds for selfish and greedy load balancing. In *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I 33*, pages 311–322. Springer, 2006.

[CGKM09] Jivitej S Chadha, Naveen Garg, Amit Kumar, and VN Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 679–684, 2009.

[CGV17] Ioannis Caragiannis, Vasilis Gkatzelis, and Cosimo Vinci. Coordination mechanisms, cost-sharing, and approximation algorithms for scheduling. In *Web and Internet Economics: 13th International Conference, WINE 2017, Bangalore, India, December 17–20, 2017, Proceedings 13*, pages 74–87. Springer, 2017.

[CK05] George Christodoulou and Elias Koutsoupias. On the price of anarchy and stability of correlated equilibria of linear congestion games. In *European Symposium on Algorithms*, pages 59–70. Springer, 2005.

[CKJ01] Jianer Chen, Iyad A Kanj, and Weijia Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001.

[CKN04] George Christodoulou, Elias Koutsoupias, and Akash Nanavati. Coordination mechanisms. In *International Colloquium on Automata, Languages, and Programming*, pages 345–357. Springer, 2004.

[CKZ01] Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 84–93, 2001.

[CL12] Yuk Hei Chan and Lap Chi Lau. On linear and semidefinite programming relaxations for hypergraph matching. *Mathematical programming*, 135(1-2):123–148, 2012.

[CLJ00] Jianer Chen, Lihua Liu, and Weijia Jia. Improvement on vertex cover for low-degree graphs. *Networks: An International Journal*, 35(4):253–259, 2000.

[CM67] Maxwell Conway and W Maxwell. Miller, theory of scheduling. *Reading: Addison Wesley*, 1967.

[CM22] José R Correa and Felipe T Muñoz. Performance guarantees of local search for minsum scheduling problems. *Mathematical Programming*, 191(2):847–869, 2022.

[CMP14] Giorgos Christodoulou, Kurt Mehlhorn, and Evangelia Pyrga. Improving the price of anarchy for selfish routing via coordination mechanisms. *Algorithmica*, 69(3):619–640, 2014.

[CP99] Jens Clausen and Michael Perregaard. On the best search strategy in parallel branch-and-bound: Best-first search versus lazy depth-first search. *Annals of Operations Research*, 90:1–17, 1999.

[CQ12] José R Correa and Maurice Queyranne. Efficiency of equilibria in restricted uniform machine scheduling with total weighted completion time as social cost. *Naval Research Logistics (NRL)*, 59(5):384–395, 2012.

[CV07] Artur Czumaj and Berthold Vöcking. Tight bounds for worst-case equilibria. *ACM Transactions On Algorithms (TALG)*, 3(1):1–17, 2007.

[Cyg13] Marek Cygan. Improved approximation for 3-dimensional matching via bounded pathwidth local search. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 509–518. IEEE Computer Society, 2013.

[DCD95] Pallab Dasgupta, P. P. Chakrabarti, and S. C. DeSarkar. A near optimal algorithm for the extended cow-path problem in the presence of relative errors. In *Foundations of Software Technology and Theoretical Computer Science*, pages 22–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[DF92] Rodney G Downey and Michael R Fellows. Fixed-parameter tractability and completeness. In *Complexity Theory: Current Research*, pages 191–225, 1992.

[DF12] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[DFKP04] Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, April 2004.

[DJ12] Nikhil R Devanur and Kamal Jain. Online matching with concave returns. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 137–144, 2012.

[DJK13] Nikhil R Devanur, Kamal Jain, and Robert D Kleinberg. Randomized primal-dual analysis of ranking for online bipartite matching. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 101–107. SIAM, 2013.

[Doe19] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. In Benjamin Doerr and Frank Neumann, editors, *Theory of evolutionary computation: Recent developments in discrete optimization*, pages 1–87. 2019.

[DS05] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

[EFFS21] Alon Eden, Michal Feldman, Amos Fiat, and Kineret Segal. An economics-based analysis of ranking for online bipartite matching. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 107–110. SIAM, 2021.

[EOJ12] Mourad El Ouali and Gerold Jäger. The b-matching problem in hypergraphs: Hardness and approximability. In *International Conference on Combinatorial Optimization and Applications*, pages 200–211. Springer, 2012.

[FHTZ22] Matthew Fahrbach, Zhiyi Huang, Runzhou Tao, and Morteza Zadimoghaddam. Edge-weighted online bipartite matching. *Journal of the ACM*, 69(6):1–35, 2022.

[FJWY22] Samuel Fiorini, Gwenael Joret, Stefan Weltge, and Yelena Yuditsky. Integer programs with bounded subdeterminants and two nonzeros per row. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 13–24. IEEE, 2022.

[FOV08] Babak Farzad, Neil Olver, and Adrian Vetta. A priority-based model of routing. *Chicago Journal of Theoretical Computer Science*, 1, 2008.

[Fre93] G.N. Frederickson. An optimal algorithm for selection in a min-heap. *Information and Computation*, 104(2):197–214, June 1993.

[FS10] Lisa Fleischer and Zoya Svitkina. Preference-constrained oriented matching. In *2010 Proceedings of the Seventh Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 66–73. SIAM, 2010.

[GGKS19] Naveen Garg, Anupam Gupta, Amit Kumar, and Sahil Singla. Non-clairvoyant precedence constrained scheduling. *arXiv preprint arXiv:1905.02133*, 2019.

[GK07] Naveen Garg and Amit Kumar. Minimizing average flow-time: Upper and lower bounds. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 603–613. IEEE, 2007.

[GKŁ97] Ervin Győri, Alexandr V Kostochka, and Tomasz Łuczak. Graphs without short odd cycles are nearly bipartite. *Discrete Mathematics*, 163(1-3):279–284, 1997.

[GKM+19] Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *60th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 26–37. IEEE, 2019.

[GKP12] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *International Workshop on Approximation and Online Algorithms*, pages 173–186. Springer, 2012.

[Gle22] Ambros M Gleixner. personal communication, November 2022.

[GLLK79] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[GM08] Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA*, volume 8, pages 982–991, 2008.

[GMUX20] Varun Gupta, Benjamin Moseley, Marc Uetz, and Qiaomin Xie. Greed works—online algorithms for unrelated machine stochastic scheduling. *Mathematics of operations research*, 45(2):497–516, 2020.

[GX16] Wayne Goddard and Honghai Xu. Fractional, circular, and defective coloring of series-parallel graphs. *Journal of Graph Theory*, 81(2):146–153, 2016.

[Hal02] Eran Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623, 2002.

[Har24] David G Harris. Dependent rounding with strong negative-correlation, and scheduling on unrelated machines to minimize completion time. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2275–2304. SIAM, 2024.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[HKT+20] Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. Fully online matching. *J. ACM*, 67(3):17:1–17:25, 2020.

[Hoc82] Dorit S Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on computing*, 11(3):555–556, 1982.

[Hoc83] Dorit S Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6(3):243–254, 1983.

[Hor73] WA Horn. Minimizing average flow time with parallel machines. *Operations research*, 21(3):846–847, 1973.

[HSS06] Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. *Computational complexity*, 15:20–39, 2006.

[HSW98] Han Hoogeveen, Petra Schuurman, and Gerhard J Woeginger. Non-approximability results for scheduling problems with minsum criteria. In *International conference on integer programming and combinatorial optimization*, pages 353–366. Springer, 1998.

[HTWZ19] Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Online vertex-weighted bipartite matching: Beating 1-1/e with random arrivals. *ACM Transactions on Algorithms (TALG)*, 15(3):1–15, 2019.

[HTWZ20] Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Fully online matching II: beating ranking and water-filling. In *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1380–1391. IEEE, 2020.

[HU11] Ruben Hoeksma and Marc Uetz. The price of anarchy for minsum related machine scheduling. In *International workshop on approximation and online algorithms*, pages 261–273. Springer, 2011.

[HZZ20] Zhiyi Huang, Qiankun Zhang, and Yuhao Zhang. Adwords in a panorama. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1416–1426. IEEE, 2020.

[IKM17] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. *Journal of the ACM (JACM)*, 65(1):1–33, 2017.

[IKMP14] Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 531–540. IEEE, 2014.

[IL23] Sungjin Im and Shi Li. Improved approximations for unrelated machine scheduling. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2917–2946. SIAM, 2023.

[ILMS09] Nicole Immorlica, Li Erran Li, Vahab S Mirrokni, and Andreas S Schulz. Coordination mechanisms for selfish scheduling. *Theoretical computer science*, 410(17):1589–1598, 2009.

[IM11] Sungjin Im and Benjamin Moseley. An online scalable algorithm for minimizing lk-norms of weighted flow time on unrelated machines. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 95–108. SIAM, 2011.

[IS20] Sungjin Im and Maryam Shadloo. Weighted completion time minimization for unrelated machines via iterative fair contention resolution*. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2790–2809. SIAM, 2020.

[Jäg23] Sven Jäger. An improved greedy algorithm for stochastic online scheduling on unrelated machines. *Discrete Optimization*, 47:100753, 2023.

[JLM25] Sven Jäger, Alexander Lindermayr, and Nicole Megow. The power of proportional fairness for non-clairvoyant scheduling under polyhedral constraints. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3901–3930. SIAM, 2025.

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[Kar05] George Karakostas. A better approximation ratio for the vertex cover problem. In *International Colloquium on Automata, Languages, and Programming*, pages 1043–1050. Springer, 2005.

[Kas25] Danish Kashaev. Selfish, local and online scheduling via vector fitting. *arXiv preprint arXiv:2505.10082*, 2025.

[KK86] Tsuyoshi Kawaguchi and Seiki Kyan. Worst case bound of an lrf schedule for the mean weighted flow-time problem. *SIAM Journal on Computing*, 15(4):1119–1129, 1986.

[KM14] Janardhan Kulkarni and Vahab Mirrokni. Robust price of anarchy bounds via lp and fenchel duality. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1030–1049. SIAM, 2014.

[KMPS09] VS Anil Kumar, Madhav V Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. A unified approach to scheduling on unrelated parallel machines. *Journal of the ACM (JACM)*, 56(5):1–31, 2009.

[KMT11] Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 587–596, 2011.

[Kol13] Konstantinos Kollias. Nonpreemptive coordination mechanisms for identical machines. *Theory of Computing Systems*, 53:424–440, 2013.

[KP94] Bala Kalyanasundaram and Kirk R. Pruhs. Constructing competitive tours from local information. *Theoretical Computer Science*, 130(1):125–138, August 1994.

[KP99] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Annual symposium on theoretical aspects of computer science*, pages 404–413. Springer, 1999.

[KP00a] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM (JACM)*, 47(4):617–643, 2000.

[KP00b] Bala Kalyanasundaram and Kirk R Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 233(1-2):319–325, 2000.

[KP09] Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *Automata, Languages and Programming, 36th Internatilonal Colloquium (ICALP), Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 508–520. Springer, 2009.

[KR08] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[KRTV13] Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *European symposium on algorithms*, pages 589–600. Springer, 2013.

[KS23] Danish Kashaev and Guido Schäfer. Round and bipartize for vertex cover approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, volume 275. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

[KST17] Christos Kalaitzis, Ola Svensson, and Jakub Tarnawski. Unrelated machine scheduling of jobs with uniform smith ratios. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2654–2669. SIAM, 2017.

[KSW86] Richard M Karp, Michael E Saks, and Avi Wigderson. On a search problem related to branch-and-bound procedures. In *27th Annual Symposium on Foundations of Computer Science*, pages 19–28, Washington, DC, 1986. IEEE Computer Society.

[KTW96] Hans Kellerer, Thomas Tautenhahn, and Gerhard J Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*, pages 418–426, 1996.

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

[KVV90] Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, 1990.

[KW14] Stefan Kratsch and Magnus Wahlström. Compression via matroids: a randomized polynomial kernel for odd cycle transversal. *ACM Transactions on Algorithms (TALG)*, 10(4):1–15, 2014.

[Las01] Jean B Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on optimization*, 11(3):796–817, 2001.

[Li24] Shi Li. Approximating unrelated machine weighted completion time using iterative rounding and computer assisted proofs. *arXiv preprint arXiv:2404.04773*, 2024.

[LKB77] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pages 343–362. Elsevier, 1977.

[LLMV20] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1859–1877. SIAM, 2020.

[LM22] Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 357–368, 2022.

[LRS11] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.

[LS99] J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, (2):173–187, May 1999.

[LV21] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM (JACM)*, 68(4):1–25, 2021.

[LZ17] Andrea Lodi and Giulia Zarpellon. On learning and branching: A survey. *TOP*, 25(2):207–236, July 2017.

[Meh13] Aranyak Mehta. Online matching and ad allocation. *Found. Trends Theor. Comput. Sci.*, 8(4):265–368, 2013.

[MJSS16] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, February 2016.

[MMS12] Nicole Megow, Kurt Mehlhorn, and Pascal Schweitzer. Online graph exploration: New results on old and new algorithms. *Theoretical Computer Science*, 463:62–72, December 2012.

[MP80] J.I. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323, 1980.

[MRST20] Yuhang Ma, Paat Rusmevichientong, Mika Sumida, and Huseyin Topaloglu. An approximation algorithm for network revenue management under nonstationary arrivals. *Oper. Res.*, 68(3):834–855, 2020.

[MS85] Burkhard Monien and Ewald Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22(1):115–123, 1985.

[MSV24] Javier Marinkovic, José A. Soto, and Victor Verdugo. Online combinatorial assignment in independence systems. In *Integer Programming and Combinatorial Optimization - 25th International Conference (IPCO)*, volume 14679 of *Lecture Notes in Computer Science*, pages 294–308. Springer, 2024.

[MSVV07] Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22–es, 2007.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: An Introduction to Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, 2005.

[MY11] Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 597–606, 2011.

[Nas24] John F Nash. Non-cooperative games. In *The Foundations of Price Theory Vol 4*, pages 329–340. Routledge, 2024.

[NR99] Rolf Niedermeier and Peter Rossmanith. Upper bounds for vertex cover further improved. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 561–570. Springer, 1999.

[NR03] Rolf Niedermeier and Peter Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *Journal of Algorithms*, 47(2):63–77, 2003.

[NT75] George L Nemhauser and Leslie Earl Trotter. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975.

[PPSV15] Andrea Pietracaprina, Geppino Pucci, Francesco Silvestri, and Fabio Vandin. Space-efficient parallel algorithms for combinatorial search problems. *Journal of Parallel and Distributed Computing*, 76:58–65, 2015.

[PSK18] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems*, 31, 2018.

[PSST22] Marco Pavone, Amin Saberi, Maximilian Schiffer, and Matt Wu Tsao. Technical note - online hypergraph matching with delays. *Oper. Res.*, 70(4):2194–2212, 2022.

[PY88] Christos Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 229–234, 1988.

[Rou15] Tim Roughgarden. Intrinsic robustness of the price of anarchy. *Journal of the ACM (JACM)*, 62(5):1–42, 2015.

[Rou21] Tim Roughgarden. *Beyond the worst-case analysis of algorithms*. Cambridge University Press, 2021.

[RS13] Mona Rahn and Guido Schäfer. Bounding the inefficiency of altruism through social contribution games. In *International Conference on Web and Internet Economics*, pages 391–404. Springer, 2013.

[RSV04] Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

[RT87] Prabhakar Raghavan and Clark D Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[Sch98] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[Sch03] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[Sch11] Uwe Schwiegelshohn. An alternative proof of the kawaguchi–kyan bound for the largest-ratio-first rule. *Operations Research Letters*, 39(4):255–259, 2011.

[SF99] Ulrike Stege and Michael Ralph Fellows. An improved fixed parameter tractable algorithm for vertex cover. *Technical report/Departement Informatik, ETH Zürich*, 318, 1999.

[Sin19] Mohit Singh. Integrality gap of the vertex cover linear programming relaxation. *Operations Research Letters*, 47(4):288–290, 2019.

[Sku01] Martin Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the ACM (JACM)*, 48(2):206–242, 2001.

[Smi56] Wayne Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[SS93] Leena M. Suhl and Uwe H. Suhl. A fast LU update for linear programming. *Annals of Operations Research*, 43(1):33–47, January 1993.

[SS95] Anand Srivastav and Peter Stangier. Weighted fractional and integral k-matching in hypergraphs. *Discrete applied mathematics*, 57(2-3):255–269, 1995.

[SS99] Jay Sethuraman and Mark S Squillante. Optimal scheduling of multiclass parallel machines. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 963–964, 1999.

[ST93] David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical programming*, 62(1):461–474, 1993.

[STZ04] Subhash Suri, Csaba D Tóth, and Yunhong Zhou. Selfish load balancing and atomic congestion games. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 188–195, 2004.

[SU11] Edward R Scheinerman and Daniel H Ullman. *Fractional graph theory: a rational approach to the theory of graphs*. Courier Corporation, 2011.

[SW00] Martin Skutella and Gerhard J Woeginger. A ptas for minimizing the total weighted completion time on identical parallel machines. *Mathematics of Operations Research*, 25(1):63–75, 2000.

[Tho06] Thomas Kamphans. *Models and Algorithms for Online Exploration and Search*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2006.

[TU24] Thorben Tröbst and Rajan Udwani. Almost tight bounds for online hypergraph matching. *Oper. Res. Lett.*, 55:107143, 2024.

[WW15] Yajun Wang and Sam Chiu-wai Wong. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *Automata, Languages, and Programming - 42nd International Collo-*

quium (ICALP), Proceedings, Part I, volume 9134 of *Lecture Notes in Computer Science*, pages 1070–1081. Springer, 2015.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227. IEEE Computer Society, 1977.

Samenvatting

De vragen die in dit proefschrift worden bestudeerd kunnen globaal als volgt worden geformuleerd: gegeven een NP-moeilijk combinatorisch optimalisatieprobleem en een suboptimale oplossing voor dit probleem – bijvoorbeeld verkregen via een efficiënt benaderingsalgoritme – wat is de kwaliteit van deze suboptimale oplossing? De kwaliteit wordt gemeten door de slechtst mogelijke verhouding, te berekenen tussen de kosten van de suboptimale oplossing en die van de optimale.

We bestuderen deze vraag in drie verschillende, maar verwante contexten. De beschouwde suboptimale oplossing kan afkomstig zijn van een standaard benaderingsalgoritme dat vooraf beschikt over alle parameters van het probleem. In dit geval beschouwen we de *approximation ratio* als maatstaf. In een *online* context worden de parameters na verloop van tijd onthult, en moet een online algoritme telkens bij elke onthulling een onomkeerbare beslissing nemen. In deze context is de relevante maatstaf de *competitive ratio*. Ten slotte kan de oplossing een stabiele uitkomst zijn die voortkomt uit de strategische interacties van spelers (bijvoorbeeld een Nash evenwicht), waarvoor de relevante maatstaf de *price of anarchy* is.

In dit proefschrift beschouwen we drie belangrijke klassen van combinatorische optimalisatieproblemen – covering, matching en scheduling – in de reeds geintroduceerde offline, online en speltheoretische modellen. Een overkoepelend thema in al onze resultaten is het gebruik van convex programmeringsrelaxaties, zoals lineaire programmering (LP) en semidefiniete programmering (SDP). We maken vaak gebruik van de kracht van dualiteit in convexe programmering om zorgvuldig gekozen duale oplossingen te construeren die we gebruiken in onze analyses en helpen bij het ontwerpen van onze algoritmen.

In Hoofdstuk 3 kijken we naar het klassieke *vertex cover* probleem en analyseren een benaderingsalgoritme in een "beyond the worst-case" model. Een belangrijke parameter is de oneven *girth*, die gedefinieerd is als de lengte van de kortste oneven cyclus in de graaf en die we gebruiken om te bepalen hoeveel de graaf van een bipartiete graaf verschilt. Met de ontwikkelde technieken bewijzen we een best mogelijke waarde van de *integrality gap* van de standaard lineaire programmeringsrelaxatie voor driekleurige grafen. Een interessante vraag is of soortgelijke technieken kunnen worden toegepast op andere combinatorische optimalisatieproblemen in verschillende beyond worst-case modellen. We geloven dat

zulke modellen een ander perspectief bieden en nieuwe inzichten kunnen geven, zelfs voor klassieke, goed bestudeerde problemen.

In Hoofdstuk 4 bestuderen we een generalisatie van het online bipartite matching probleem en beschouwen we hypergrafen. We concentreren ons op het driedimensionale probleem waarbij knopen na verloop van tijd bekend worden gemaakt, en bewijzen dat de best mogelijke competitieve ratio $(e-1)/(e+1)$ is voor het fractionele probleem. Een interessante open vraag is nog steeds of de grens van $1/3$ voor het integrale probleem, die eenvoudig wordt bereikt door een naïef algoritme, verbeterd kan worden. We geloven dat dit probleem aanzienlijk moeilijker is dan voor bipartiete grafen, en dat geavanceerde correlated/dependent rounding technieken nodig zullen zijn voor verdere vooruitgang.

In Hoofdstukken 5 en 6 ontwikkelen we een *dual fitting* framework gebaseerd op een enkel semidefinite programma waarmee we nauwkeurig kunnen analyseren de price of anarchy voor spellen, de approximation ratio van lokale zoekalgoritmes en de competitive ratio van online algoritmes voor verschillende machine scheduling problemen. Met deze techniek kunnen we op eenvoudige en uniforme wijze bewijzen geven voor veel belangrijke resultaten. Deze omvatten onder andere de analyse van de price of anarchy voor het unrelated machine scheduling probleem $R||\sum w_jC_j$, de best bekende deterministische en gerandomiseerde coördinatiemechanismen voor dit probleem, evenals het best bekende combinatorische offline benaderingsalgoritme. In de speltheoretische context herleiden we ook de price of anarchy van gewogen affiene congestiespellen en de pure price of anarchy van scheduling op parallelle machines. In de online setting herleiden we de best bekende deterministische en gerandomiseerde algoritmen voor het online load balancing probleem op unrelated machines en presenteren we een best mogelijk fractioneel algoritme. Het kan interessant zijn om deze techniek toe te passen op andere allocatie problemen waarvan de optimale oplossing gemodelleerd kan worden als een binair kwadratisch programma.

In Hoofdstuk 7 bestuderen we het explorable heap selection probleem. Het doel is om de n^{de} kleinste waarde te vinden in een binaire heap, en de complexiteit van het algoritme wordt gemeten aan de hand van de totale afstand die in de boom wordt afgelegd. Wij presenteren een nieuw gerandomiseerd algoritme dat de best bekende looptijd verbetert, zij het ten koste van een iets groter geheugengebruik.

Curriculum Vitae

Danish Kashaev was born on the 3rd of April 1997 in Helsinki, Finland. He is of tatar origin and grew up in Geneva, Switzerland since 2002. He obtained his bachelor's degree in mathematics at the University of Geneva in 2018, while having spent the third academic year at the University of California, Santa Barbara as an exchange student. He then started a master's degree in mathematics at ETH Zurich, which he completed in 2021. A few months after the completion of his master's degree, he joined the Networks and Optimization group at CWI Amsterdam as a doctoral student. His two favourite hobbies since childhood are playing tennis and the piano.

A combinatorial optimization problem involves finding a solution which minimizes or maximizes an objective function among a very large set of potential feasible solutions. Many interesting optimization problems are known to be NP-hard, meaning that it is unlikely that an efficient (or polynomial-time) algorithm exists to find an optimal solution for such problems. Naturally, it becomes interesting to study efficient algorithms which find suboptimal solutions, whose quality can nevertheless still be proven to be close to the optimal one.

How close can the cost of a given suboptimal solution get to the best possible one? This thesis studies this question in different settings, where such a solution is either the output of a classical approximation algorithm, the output of an online algorithm operating in a more restrictive computational model, or a Nash equilibrium arising in a game-theoretic context. It explores both upper and lower bounds on this question for three important classes of combinatorial optimization problems and introduces new techniques for proving tight results using tools such as linear programming and semidefinite programming duality.

Danish Kashaev (1997) received a Bachelor's degree in Mathematics from the University of Geneva in 2018 and a Master's degree in Mathematics from ETH Zurich in 2021. The research for this PhD thesis was conducted at the Centrum Wiskunde & Informatica (CWI) in Amsterdam between 2021 and 2025.

ISBN: 978-94-6536-004-1