Applications of Strong Convex Relaxations to Allocation Problems

THÈSE N^{O} 7468 (2017)

PRÉSENTÉE LE 10 MARS 2017

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE THÉORIE DU CALCUL 2
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Christos KALAITZIS

acceptée sur proposition du jury:

Prof. B. Rimoldi, président du jury
Prof. O. N. A. Svensson, directeur de thèse
Prof. N. Bansal, rapporteur
Prof. M. Cygan, rapporteur
Prof. F. Eisenbrand, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2017

. . . nor deemed I that thy decrees were of such force,
that a mortal could override the unwritten and unfailing statutes of heaven.
For their life is not of today or yesterday,
but from all time, and no man knows when they were first put forth.
— Sophocles, Antigone

To my parents,
Giorgos and Athanasia

Acknowledgements

As my journey towards becoming a doctor comes to an end, I feel the need to thank all the people that enabled and facilitated this journey. Nothing important can ever be achieved without the help of others, and acknowledging this help is the least an honest man could and should do.

First of all, I want to express my deep gratitude to Ola for being a great advisor and friend. Nobody influenced my formation as a scientist more than him, and I could not be more appreciative of the way he did it. While his energy, drive and enthusiasm are extremely contagious, even more important is the fact that through his own actions Ola sets an example, one that compels you to always try harder. Somehow, I feel that this strive for excellence is the most important lesson Ola ever taught me, and I will certainly try to render it the most well-learned one as well.

Next, I would like to thank the members of my jury Prof. Nikhil Bansal, Prof. Marek Cygan and Prof. Friedrich Eisenbrand, and the president Prof. Bixio Rimoldi, for their valuable comments and remarks on my thesis, but also for being part of the most important event of my academic life so far.

I would also like to thank all of my co-authors for enabling me to grow as an academic through our work together: Chidambaram Annamalai, Aleksander Mądry, Alantha Newman, Lukáš Poláček and Jakub Tarnawski. Our joint efforts were one of the most important and defining aspects of my PhD life, and the lessons I learned from you will accompany me for the rest of my life.

Furthermore, I would like to mention how much fun it was being in the same lab as such amazing people: Abbas, Aida, Amir, Ashkan, Damian, Farah, Hyung-Chan, Justin, Sangxia, and Slobodan! All these hours we spent together were some of the most fun and creative times of my life, and many of the harder parts of going through my PhD studies were made much easier thanks to you. I do not think I could have made it to the end without you, and for this I thank you! Special thanks should also go to Chantal, who made so many things possible for us, without ever dropping her smile!

Throughout my stay in Lausanne, I made a lot of new and extraordinary friends, who are directly responsible for a big part of the good times I had here. The following list, though probably meager and certainly incomplete, is the only acknowledgement I could put on a piece of paper: Manolis, Alexandros, Pavlos, Anastasia, Stefanos, Onur, Kuveli, Eleni, Natalia, Ioannis, Panagiotis, Marco, Andrea, Jean, Fay, Serj, Rafah, Marwa, Giorgos C., Giorgos Ps., Giorgos Pr., Eleni, Manos, Iliana, Apostolis, Javier, Iraklis, Stavros, Goran, Vasilis, Stella, Panagiotis, Matt,

Katerina, Ali, Alfonso, Mani, Aris, Rajai, Abdullah and all the people I forget, the memories of all the beer glasses we emptied together shall never wash away!
Next, I want to thank my friends from home for all their ((very) harsh) love and (questionable) support throughout my time as a PhD student, but even more throughout its darkest, fairly immobile part: Yannis, Sotiris, Thanasis, Giorgos, Vasilis, Elias, Elias, Nikolas[1] and Nikos, thank you for successfully lifting my spirits by unsuccessfully trying to bring them to absolute zero!
Finally, I would like to thank the three people that made writing this acknowledgement go from a dream to a possibility and then to a reality: my brother Dimitris, whom I love more than I will ever say, and my parents Giorgos and Athanasia, whom I love more than I could ever say. From any point of view, you really are my best teachers ever, and there can never be a way to say how grateful I am to be your son and brother.

[1] "I'm bringing the wheel-saw to break the cast, then you can walk again!"

Abstract

Approximation algorithms are a commonly used tool for designing efficient algorithmic solutions for intractable problems, at the expense of the quality of the output solution. A prominent technique for designing such algorithms is the use of Linear Programming (LP) relaxations. An optimal solution to such a relaxation provides a bound on the objective value of the optimal integral solution, to which we compare the integral solution we return.

In this context, when studying a specific problem, two natural questions often arise: What is a strong LP relaxation for this problem, and how can we exploit it? Over the course of the past few decades, a significant amount of effort has been expended by the research community in order to answer these questions for a variety of interesting intractable problems. Although there exist multiple problems for which we have designed LP relaxations that achieve best-possible guarantees, there still exist numerous problems for which we either have no strong LP relaxations, or do not know how to use them. The main focus of this thesis is extending our understanding of such strong relaxations.

We focus on designing good approximation algorithms for certain allocation problems, by employing a class of strong LP relaxations, called configuration-LPs. For many such allocation problems, the best-known results are derived by using simple and natural LP relaxations, whereas configuration-LPs have been used successfully on several occasions in order to break pre-existing barriers set by weaker relaxations. However, our understanding of configuration-LPs is far from complete for many problems. Therefore, understanding and using these relaxations to the farthest extent possible is a quite intriguing question. Answering this question could result in improved approximation algorithms for a wide variety of allocation problems.

The first problem we address in this thesis is the restricted max-min fair allocation problem. Prior to our work, the best known result [19] provided an $\Omega(1)$-approximation that ran in polynomial time. Also, it was known [2] how to estimate the value of an optimal solution to the problem within a factor of $\frac{1}{4+\epsilon}$, for any $\epsilon > 0$, by solving the corresponding configuration-LP. Our first contribution in this thesis is the design of a $1/13$-approximation algorithm for the problem, using the configuration-LP. Specifically, although our algorithm is fully combinatorial, it consists of a local-search procedure that is guaranteed to succeed only when the configuration-LP is feasible. In order to establish the correctness and running time of the algorithm, it is crucial to use the configuration-LP in our analysis.

The second problem we study is the scheduling of jobs on unrelated machines in order to minimize the sum of weighted completion times. For this problem, the best known approxi-

mation algorithm [5] achieves a ratio of $3/2-\epsilon$, for some small $\epsilon > 0$. Our second contribution in this thesis is the improvement of this ratio to $\frac{1+\sqrt{2}}{2}+\varepsilon$, for any $\varepsilon > 0$, for the special case of the problem where the jobs have uniform Smith ratios. To achieve this ratio, we design a randomized rounding algorithm that rounds solutions to the corresponding configuration-LP. Through a careful examination of the distribution this randomized algorithm outputs, we identify the one that maximizes the approximation ratio, and we then upper bound the ratio this worst-case distribution exhibits by $\frac{1+\sqrt{2}}{2}+\varepsilon$. Finally, we remark that our analysis of the rounding algorithm is tight.

Key words: Approximation Algorithms, Linear Programming, **NP**-hard problems, Allocation Problems

Résumé

Les algorithmes d'approximation sont des instruments souvent utilisés pour concevoir des solutions algorithmiques efficaces à des problèmes intraitables, au détriment de la qualité de la solution produite. L'utilisation des relaxations de Programmation Linéaire (LP) est une technique importante pour concevoir de tels algorithmes. Une solution optimale à une telle relaxation nous donne une limite inférieure de la valeur objective d'une solution intégrale optimale, limite à laquelle nous comparons la solution intégrale trouvée.

Quand on étudie un problème spécifique, on fait face à deux questions : (1) Quelle est la relaxation LP la plus puissante pour ce problème ? et, (2) comment mieux l'utiliser ? Durant les dernières décennies, la communauté de recherche a déployé beaucoup d'efforts pour résoudre plusieurs problèmes intraitables intéressants. Bien qu'il y ait beaucoup de problèmes pour lesquels on a conçu des relaxations LP qui fournissent des algorithmes avec les meilleures garanties possibles, il y a de nombreux problèmes pour lesquels on n'a pas de relaxations LP assez puissantes, ou on a de telles relaxations, mais on ne sait pas comment les utiliser au mieux. L'objectif principal de cette thèse est de mieux comprendre de telles relaxations puissantes.

Nous nous concentrons sur la conception d'algorithmes d'approximation efficaces pour des problèmes d'allocation, en utilisant une classe de puissantes relaxations LP, nommées configuration-LPs. Pour beaucoup de problèmes d'allocation, les algorithmes les plus puissants sont conçus en utilisant des relaxations LP simples et naturelles, alors que des configuration-LPs ont été utilisées à beaucoup d'occasions pour dépasser les limites fixées par des relaxations LP moins fortes. Cependant, notre compréhension des configuration-LPs est loin d'être parfaite pour de nombreux problèmes. Par conséquent, comprendre parfaitement et utiliser ces relaxations reste une question importante. Répondre à cette question nous permettra peut-être de concevoir des algorithmes d'approximation plus efficaces pour beaucoup de problèmes d'allocation.

Le premier problème considéré dans cette thèse est le problème de restricted max-min fair allocation. Avant notre travail, le meilleur algorithme connu [19] avait une garantie d'approximation de $\Omega(1)$. De plus, on savait [2] comment estimer la valeur d'une solution optimale avec une garantie de $\frac{1}{4+\epsilon}$, pour toute $\epsilon > 0$, en utilisant la configuration-LP. La première contribution de cette thèse est la conception d'un algorithme d'approximation qui s'exécute en temps polynomial, et qui a une garantie d'approximation de $1/13$. Bien que notre algorithme soit combinatoire, il consiste en une procédure local-search qui s'exécute avec succès quand la configuration-LP a des solutions faisables. Pour prouver que l'algorithme fonctionne correcte-

ment et s'exécute en temps polynomial, l'utilisation de la configuration-LP dans notre analyse est très importante.
Le second problème considéré est le problème de job-scheduling on unrelated machines dont l'objectif est de minimiser la somme pondérée des temps d'achèvement. Pour ce problème, le meilleur algorithme connu [5] a une garantie d'approximation de $3/2-\epsilon$, pour un petit ϵ. La seconde contribution de cette thèse est la conception d'un algorithme avec une garantie d'approximation de $\frac{1+\sqrt{2}}{2}+\varepsilon$, pour toute $\varepsilon > 0$, dans le cas de travaux ayant des Smith-ratios uniformes. Pour atteindre cette garantie d'approximation, nous concevons un algorithme randomisé qui arrondit des solutions de la configuration-LP. Pour analyser cet algorithme, nous examinons toutes les distributions des travaux que l'algorithme peut produire, et nous trouvons la distribution la moins efficace. Puis, nous calculons une limite supérieure de la garantie d'approximation de cette distribution, qui est $\frac{1+\sqrt{2}}{2}+\varepsilon$.

Mots clefs : Algorithmes d'Approximation, Programmation Linéaire, Problèmes **NP**-hard, Problèmes d'Allocation

Contents

List of Figures

List of Tables

1 Introduction

Determining what tasks we can perform by using machines has always been a driving question for computer science. Recent technological advances have enabled us to perform computations at a huge, ever increasing, scale. Computing machines are becoming stronger and better interconnected every day, enabling us to carry out complex and cumbersome tasks, that we would not have dared to dream of even a few decades ago. The nature, however, of the computations we can perform, as well as the level of efficiency in doing so, has been a question on its own. In fact, it was a question even before the design of the first digital electronic computers, that have changed almost every aspect of our lives and the levels at which this question is valid are multiple. Answering some of these questions, by providing algorithmic solutions for specific problems, is the main focus of this thesis.

On the most fundamental (and perhaps philosophical) level, computer scientists have been reassured by the universally accepted Church-Turing Thesis [46]. This is an informal proposition that states that every function, which can be computed by a human who is supplied with infinite resources and a clear set of instructions, can also be computed by a Turing Machine, a formal expression of the algorithmic concept. This proposition is oblivious, however, to the resources we need in order to compute a specific function or to solve a specific problem. Humans will always have a limited supply of time; computers will always have a limited supply of memory and power; computer networks will always have a limited bandwidth; and, in general, the tasks we can perform on a computer will always be limited by our available resources. Understanding some of the limitations on the problems that can be solved algorithmically is one of our goals in this thesis.

The fact that computational resources are limited brings the problem of designing *efficient* algorithms under the spotlight. The most common measure of efficiency of an algorithm, and in fact the one around which this thesis will revolve, is time. However, what constitutes a time-efficient algorithm varies according to context. Depending on the efficiency perspective, such algorithms were designed to have, on average, good running times on all possible inputs, or to perform well on input instances that often appear in practice.

Our approach is to design algorithms that exhibit good worst-case behaviour, i.e., algorithms that provably terminate after a certain (preferably small) amount of time, depending on the input size. One of the most important steps in the process of designing such an algorithm is understanding the combinatorial structure of the underlying problem. This understanding facilitates both the actual design of the algorithm and its analysis. Taking into account time restrictions, the class of problems that we can efficiently solve on a computer, with provable guarantees, is vastly restricted. In fact, one of the most popular perspectives on what problems we can efficiently solve is given by the Cobham-Edmonds thesis [13, 16]: the class of efficiently solvable problems is given by the class of problems belonging to **P**.

Fortunately, the class of problems, for which we have exact polynomial-time algorithms, is large and contains quite a few very relevant problems in practice. Some of these problems we understand in great depth, to an extent that we have designed theoretically optimal algorithms that also perform well in practical settings. However, these do not represent all the problems we would like to solve. Quite often, we encounter problems that we know we will never be able to solve in polynomial time, or problems that we do not know whether we can solve in polynomial time. Representatives of the latter are the problems we call **NP**-complete, which is arguably the most natural (and interesting) superset of **P** for which we do not know whether it admits efficient algorithms.

As **NP**-hard optimization problems often arise in practice, we are compelled to address them and to design good algorithms for them. In the absence of a conclusive positive answer to the **P=NP** question, we are forced to make certain compromises when designing such algorithms. In general, there are three desiderata when designing algorithms for **NP**-hard optimization problems:

a. Optimality: the designed algorithm outputs an optimal solution.

b. Universality: the designed algorithm performs correctly on all possible inputs.

c. Efficiency: the designed algorithm provably terminates in polynomial time.

Designing an algorithm that satisfies all of the above for an **NP**-hard optimization problem would be equivalent to proving **P=NP**. Therefore, there are three general trends in designing algorithms for such problems: these trends depend precisely on which condition we choose to relax. The scope of this thesis is restricted to designing algorithms for which we have relaxed the optimality criterion. Instead of outputting an optimal solution, such algorithms run in polynomial time at the expense of outputting a suboptimal solution. Typically, the objective value is provably within a multiplicative factor away from the optimal one; such algorithms are called *approximation algorithms*.

1.1 Approximation Algorithms

In the course of designing an approximation algorithm for an **NP**-hard optimization problem, we focus on two points: designing an algorithm that terminates in polynomial time, and providing a formal proof that this algorithm always outputs a solution whose value is within some factor of the optimum; this factor will be the *approximation factor* of the algorithm:

Definition 1.1.1. *Let $\mathscr{I}$ be the set of input instances of an optimization problem Π, and let Π_I be the optimal value of $I \in \mathscr{I}$. Given an algorithm $\mathscr{A}$, let $\mathscr{A}_I$ be the value of the solution $\mathscr{A}$ outputs on input $I \in \mathscr{I}$. If Π is a maximization problem, the approximation factor $\rho_{\mathscr{A}}$ of $\mathscr{A}$ is*

$$\rho_{\mathscr{A}} = \inf_{I \in \mathscr{I}} \frac{\mathscr{A}_I}{\Pi_I},$$

whereas if Π is a minimization problem, the approximation factor $\rho_{\mathscr{A}}$ of $\mathscr{A}$ is

$$\rho_{\mathscr{A}} = \sup_{I \in \mathscr{I}} \frac{\mathscr{A}_I}{\Pi_I}.$$

The above definition compares the value of an optimal solution to the one the approximation algorithm outputs; notice that the approximation ratio is larger than 1 for minimization problems, and smaller than 1 for maximization problems. As computing the optimal value of every possible instance that could be the input to our algorithm is hard, one technique that is commonly used in order to analyze the approximation ratio of a given algorithm is to compute a weaker, but easier to find, bound on the optimal value. Given this bound, we then compare it to the output value in order to estimate the approximation ratio of our given algorithm. It is quite often the case that the choice of this bound is dictated by the original design of our approximation algorithm. Consider the following example:

Example. Consider the maximum knapsack problem: Given a knapsack of size W and a set of n items, where item i has value v_i and weight $w_i \leq W$, find a subset S of items such that $\sum_{i \in S} w_i \leq W$ and such that $\sum_{i \in S} v_i$ is maximized.

Let $r_i = v_i / w_i$, for any item i. Now consider the following algorithm: Sort the items in non-increasing order of r_i, and let $k = \arg\max_i \sum_{j \leq i} w_j \leq W$. Out of the two sets $\{1, ..., k\}$ and $\{k+1\}$, output the one with highest total value.

Let us now prove that the above algorithm achieves an approximation ratio of $1/2$. Clearly, the output set has value at least $\sum_{i \leq k+1} v_i/2$. Now, it suffices to prove that $\sum_{i \leq k+1} v_i \geq \sum_{i \in S^*} v_i$, where S^* is an optimal solution to our instance. To see this, first observe the two following facts:

- $r_{k+1} \geq r_i$, for any $i \in S^* \setminus \{1, ..., k+1\}$, since the items are sorted in non-increasing order of r_i.

- $\sum_{i\in\{1,\ldots,k+1\}\setminus S^*} w_i \geq \sum_{i\in S^*\setminus\{1,\ldots,k+1\}} w_i$, since $\sum_{i\in\{1,\ldots,k+1\}} w_i \geq W \geq \sum_{i\in S^*} w_i$.

These two facts combined imply that

$$\sum_{i\in S^*\setminus\{1,\ldots,k+1\}} r_i w_i \leq \sum_{i\in S^*\setminus\{1,\ldots,k+1\}} r_{k+1} w_i \leq \sum_{i\in\{1,\ldots,k+1\}\setminus S^*} r_{k+1} w_i \leq \sum_{i\in\{1,\ldots,k+1\}\setminus S^*} r_i w_i$$

which directly implies that $\sum_{i\leq k+1} v_i \geq \sum_{i\in S^*} v_i$. □

So, in the above example we used $\sum_{i\leq k+1} v_i$ as a bound on the instance's optimal value. In general, the better such bound we use when we design and analyze an approximation algorithm, the better the algorithm's approximation ratio will be. Often, we are able to come up with good combinatorial bounds (such as the above), that give rise to combinatorial approximation algorithms. In contrast, there exists a whole family of approximation algorithms that rely on bounds that derived from *linear programming relaxations*; such algorithms will be the focal point of this thesis.

1.2 Linear Programming and Approximation

The key fact behind the use of linear programming in the design of approximation algorithms is the following simple observation: In many cases, given a maximization problem Π (the approach for minimization problems is analogous), we are able to express the problem of finding an optimal solution to an instance I of Π as the problem of finding an optimal solution to an appropriately defined integer linear program (ILP):

$$\begin{aligned} \max \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned}$$

where x is a vector of n decision variables that correspond to the solution to the problem, c is an appropriately defined weight vector, and A and b define a linear inequality system that correspond to the constraints of the problem; clearly, n, A, b and c depend on I. Given such a formulation, the optimum of I is equal to the optimum of the ILP, hence solving the ILP would provide us with an optimal solution. Solving ILP-s is **NP**-hard in general; however, given such an ILP, we can relax the integrality constraint to obtain the following linear program (LP):

$$\begin{aligned} \max \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \in \mathbb{R}^n \end{aligned}$$

which we know how to solve in polynomial time (using, e.g., the ellipsoid method [29]). Furthermore, as the set of feasible solutions to the ILP is a subset of the set of feasible solutions to the LP, the optimum of the LP is at least that of the ILP, hence the optimum of the LP will

serve as the bound to which we will compare our solution. Hence, if we are given such an LP formulation, one general strategy to design an approximation algorithm is the following: We solve the LP to retrieve a fractional optimal solution x^*, then round x^* to an integral solution $\hat{x}$ in polynomial time, and output $\hat{x}$. The ratio of the value of the rounded solution over that of the optimal fractional solution will designate the approximation factor; in other words, our approximation factor ρ will be

$$\rho = \inf_{I \in \mathscr{I}} \frac{c^T \hat{x}}{c^T x^*}$$

Now, the following question arises: What is the best approximation ratio ρ we can achieve using the above strategy? To begin with, it cannot be better than

$$\inf_{I \in \mathscr{I}} \frac{c^T \hat{x}^*}{c^T x^*}$$

where $\hat{x}^*$ is the optimal integral solution to our ILP. The above ratio is called the *integrality gap* of our relaxation. In some sense, the integrality gap is a measure of how good a relaxation is, as it places an absolute bound on the performance of any rounding algorithm for a given relaxation. Therefore, determining it will often be as important as coming up with an actual polynomial-time rounding scheme. Similarly, coming up with relaxations that have a good integrality gap will often be as important as devising an appropriate rounding scheme.

To illustrate the above concepts, we provide a small example:

Example. Consider the following ILP for the maximum knapsack problem:

$$\begin{aligned} \max \quad & \sum_{i \in [n]} x_i v_i & & \\ \text{subject to} \quad & \sum_{i \in [n]} x_i w_i & \leq W & \\ & x_i \in \{0,1\} & & \forall i \in [n] \end{aligned}$$

where x_i is a variable that should be set to 1 if we pick item i, and 0 otherwise, By relaxing the integrality constraint, we get the following LP formulation:

$$\begin{aligned} \max \quad & \sum_{i \in [n]} x_i v_i & & \\ \text{subject to} \quad & \sum_{i \in [n]} x_i w_i & \leq W & \\ & 0 \leq x_i & \leq 1 & \forall i \in [n] \end{aligned}$$

For simplicity, let us assume the value/weight ratios $r_i = v_i / w_i$ are unique. Let x^* be an optimal solution to the LP. First, we observe that there exists at most one variable in x^* that is not 0 or 1. To see this, assume there exist two non-integral variables x_i and x_j; then, due to the uniqueness of the value/weight ratios, increasing the variable with the larger ratio and decreasing the one with the smaller ratio will produce a feasible solution with a better

objective value, which is a contradiction[1].

If there is no fractional variable in x^*, we output the set of items whose corresponding variable is set to 1. Otherwise, let $S \subseteq [n]$ be the set of variables that are set to 1, and let x_j be the variable that is non-integral. Our rounding algorithm is the following: out of the two sets S and $\{j\}$, output the one with the larger total value. Since $\max\{v_j, \sum_{i \in S} v_i\} \geq \sum_{i \in [n]} x_i^* v_i/2$, the above rounding algorithm constitutes an $1/2$-approximation algorithm. Consequently, we have also proved that the integrality gap of the above relaxation is at least $1/2$. □

1.3 Allocation Problems

Let us now move on to the next focal point of this thesis: the study of allocation problems. As the name suggests, we typically use this term to refer to problems where we are required to distribute indivisible items to multiple entities. In doing so, our goal will be to optimize an objective function that expresses some notion of balance or fairness.

Such problems often arise in practice, and their practical relevance motivates us to study them. One prime example of such problems is balancing the load of processing jobs among multiple processors/machines; there, we could have different notions of balancing, such as ensuring that no machine is overloaded, or that no machine is left with too little load. Another setting in which such problems arise is conducting combinatorial auctions, where we aim to distribute items among agents such that the average happiness of the agents is maximized. Similarly, another relevant setting is that of performing budgeted allocations, i.e., allocating items to agents that pay for them, but under the constraint that no agent can pay more than his prespecified budget.

Such problems often appear to be **NP**-hard, and a natural approach to solving them is to employ linear relaxations. However, as we will see later on, it is often the case that many natural relaxations have inferior integrality gaps, and that we will have to use less natural ones in order to achieve good approximation guarantees.

1.3.1 LP Relaxations for Allocation Problems

A typical approach to designing an approximation algorithm for a given allocation problem is to use a fairly natural class of LP relaxations, called assignment-LPs. This is a class of basic LP relaxations that uses variables for the decision of allocating a single item to a single entity, and then implements a set of basic constraints that force this allocation to be consistent.

For most of the classic allocation problems we are interested in, the assignment-LP has been fully understood, in the sense that we know its exact integrality gap, and have rounding

[1] Observe that we only used the uniqueness of ratios to prove that there exists at most one fractional variable in x; instead of making this assumption, we could have achieved the same by assuming x is a so-called *extreme-point solution*.

algorithms that achieve it. In order to achieve improved approximation guarantees, one popular technique is to use a class of strong LP relaxations, called configuration-LPs. Their main difference with assignment-LPs is that they employ variables that encode the decision of allocating a *set of items* to a single entity, instead of just a single item. Contrary to the class of assignment-LPs, our understanding of these stronger relaxations is far from complete. Understanding how we can apply these relaxations to the design of better approximation algorithms is one of the main questions that we address in this thesis.

Let us give an overview of what we know about these relaxations, for some fundamental allocation problems (the definitions of these problems appear in Appendix B):

	ρ	A-LP (tight)	C-LP (positive)	C-LP (negative)
Restricted min-max allocation	2 [31]	2 [31]	11/6 [24]	3/2 (implied by [31])
Generalized assignment problem	$1-1/e+\epsilon$ [15]	1/2 [11, 39]	$1-1/e+\epsilon$ [15]	4/5 [15]
Maximum budgeted allocation	$3/4+\epsilon$ [25]	3/4 [43, 10]	$3/4+\epsilon$ [25]	0.828 [26]
Bin packing	$O(\log n)$[34] (additive)	2 (folklore)	$O(\log n)$ [34] (additive)	1 (additive) (folklore)
Restricted max-min allocation	$\Omega(1)$ [14]	unbounded [6]	1/4 [3]	1/2 [9]
Min-sum of weighted completion times scheduling	$3/2-\epsilon$ [5]	$3/2^{\star}$ [5, 38, 40]	$3/2-\epsilon$ [5]	$1+\epsilon$ (implied by [41])

Table 1.1: The first column contains the names of the problems, the second column contains our best-known approximation guarantee, the third column contains our best-known bound on the integrality gap of the assignment-LP (the displayed bounds are all tight), the fourth one that of the configuration-LP, and the fifth one the best negative bound we have on the integrality gap of the configuration-LP.
$\star$: Here, there is no natural A-LP relaxation; instead, the ratio refers to two of the more natural convex relaxations.

Inspecting the above table, the following three patterns emerge:

- For all the mentioned problems, we have tight bounds on the integrality gap of the corresponding natural relaxations, which implies that our understanding of such relaxations is complete in these cases.

- For all of these problems, the integrality gap of the configuration-LP is at least as good as the best-known approximation guarantee, and even better in some cases.

- Our understanding of the configuration-LP is complete for none of the problems we mention.

Our main focus in this thesis is to study how the configuration-LP for such problems can be used, in order to obtain improved approximation guarantees. This focus is set on the last two problems of the above table: the restricted max-min allocation problem and the min weighted completion time problem with uniform Smith ratios.

1.4 Our Contributions

In this section, we will conduct a brief overview of the main contributions of this thesis.

1.4.1 Restricted Max-Min Fair Allocation

In Chapter 3, we study the restricted max-min fair allocation problem. The problem is formally defined as follows: we are given a set of agents $\mathscr{P}$, and a set of items $\mathscr{R}$, and a value $v_{ij} \in \{v_j, 0\}$ for each agent i and item j. We want to assign a set of items S_i to each agent i, where $S_i \cap S_j = \emptyset$ for all agents i and j, such that we maximize $\min_{i \in \mathscr{P}} \sum_{j \in S_i} v_{ij}$.

For this problem, we develop a local-search algorithm, that builds upon previous approaches [4, 32] and achieves an $1/13$-approximation ratio, which improves upon the previously best known $\Omega(1)$-approximation ratio [19, 14]. Although the algorithm is purely combinatorial, its analysis is based on the strong configuration-LP relaxation for the problem. This LP relaxation serves as the bound towards which we compare our tentative estimate of the optimal value; if the local search algorithm fails, it means that the configuration-LP is infeasible for our current estimate, that we subsequently update and reiterate. In order to prove polynomial-time termination, we develop new tools, such as greedy agents and lazy updates, for conducting our local search. These tools enable us considerably accelerate the local search, compared to previous approaches.

Our results for the restricted max-min fair allocation problem were the result of joint work with Chidambaram Annamalai and Ola Svensson; this work was published in SODA 2015 [1].

1.4.2 Scheduling Jobs with Uniform Smith Ratios on Unrelated Machines

In Chapter 4, we study the min sum of weighted completion times with uniform Smith ratios. The problem is formally defined as follows: We are given a set of machines $\mathscr{M}$, a set of jobs $\mathscr{J}$, a processing time p_{ij} for each $i \in \mathscr{M}$ and $j \in \mathscr{J}$, and a weight w_j for each $j \in \mathscr{J}$, such that $p_{ij} \in \{w_j, \infty\}$ for all $j \in \mathscr{J}$ and $i \in \mathscr{M}$. Our goal is to find an assignment of jobs to machines, and a schedule of the jobs assigned to each machine, such that we minimize the sum of weighted completion times $\sum_{j \in \mathscr{J}} w_j C_j$, where C_j is the completion time of job j under the

constructed schedule.

For this problem, we prove that the strong configuration-LP has an integrality gap of $\frac{1+\sqrt{2}}{2}$, by designing a polynomial-time rounding algorithm. For this special case of the min weighted completion time problem, our result improves upon the best-known approximation ratio of $3/2-\epsilon$ [5]. Our approach towards solving this problem relies on using a randomized rounding algorithm that achieves a strong concentration of the number of jobs assigned on each machine, by defining ranges of job sizes, and then randomly picking one job from each such range. Our analysis relies on finding, for each machine, the distribution of the assigned jobs that satisfies the above constraint and maximizes the expected cost; in other words, we find the *worst-case output distribution*. Then, we apply a sequence of transformations on this distribution, in order to reach a specific type of distributions whose cost we can analyze and whose cost is an upper bound on the cost of the original worst-case distribution. We remark that our analysis is tight, i.e., there exist instances for which the ratio achieved by the rounding algorithm is $\frac{1+\sqrt{2}}{2}$.

Our results for min sum of weighted completion times scheduling with uniform Smith ratios were the result of joint work with Ola Svensson and Jakub Tarnawski. This work will be published in SODA 2017 [27].

2 Convex Relaxations for Allocation Problems

In this chapter, we introduce and discuss the LP relaxations we will study throughout this thesis. In order to describe the intuition behind these relaxations, let us go back to the basic choices we made when designing a solution to an instance of an allocation problem.

As we have already stated, when faced with such a problem, we aim to distribute items among different entities. This fact hints at the following way to formulate LP relaxations for such problems: We introduce a decision variable x_{ij} for every decision of the form "item j is assigned to i". Then, we proceed to express the problem constraints with respect to these variables. Such relaxations are typically called assignment-LPs, and using them is often the first step towards designing an approximation algorithm for an allocation problem. In many cases, these relaxations lead to quite good algorithms, and often they produce the best algorithms we know for specific problems (a notable example is the problem of minimizing the makespan on unrelated machines).

Indeed, we have a complete understanding of such relaxations for some of the most interesting allocation problems; specifically, this means that for these problems we cannot achieve improved approximation guarantees by using these relaxations. One way to bypass this problem is to add more valid constraints to our relaxation, in the hope that these constraints cut out the fractional solutions that exhibit the worst integrality gap: This process could be done by looking into a specific problem and its inherent structure, or in a more general way, by applying techniques such as LP/SDP hierarchies, that constitute an automated way to reinforce a given relaxation.

Another approach that is sometimes available, instead of considering the decision of allocating a specific item to some entity as a variable of our relaxation, is to consider the following question: Should some entity i be receiving precisely the set of items C? This decision is naturally translated to a decision variable y_{iC}, and the resulting LP relaxation, along with its consistency constraints, is now a configuration-LP. Although there are many aspects of these relaxations that we should and will discuss, perhaps the most appealing one is the following: Given a fractional solution to a configuration-LP, when we focus on some entity i, we are

given a local distribution over feasible integral assignments for i. Even though we cannot jointly sample from these distributions for all entities, estimating the integrality gap of such a relaxation amounts to measuring how much we lose in terms of our objective function by trying to output joint distributions over assignments for all entities.

Following the above discussion, in the next two sections we introduce the LP relaxations we will use for the two problems we study in this thesis.

2.1 Configuration-LP for the Restricted Max-Min Fair Allocation Problem

We begin by defining the configuration-LP for the restricted max-min fair allocation problem. First of all, this LP relaxation is what we call a *feasibility LP*, i.e., an LP without an objective function, that we solve only to check if it admits a feasible solution. Specifically, let τ be an estimate on the optimal value of our input instance. For any $i \in \mathscr{P}$, we define $\mathscr{C}(i,\tau) = \{C \subseteq \mathscr{R} : \sum_{j \in C} v_{ij} \geq \tau\}$ to be the set of configurations (i.e., sets of items) that have total value at least τ for i. Then, the configuration-LP for τ (abbreviated as C-LP(τ)) has a decision variable x_{iC} for all $i \in \mathscr{P}$ and $C \in \mathscr{C}(i,\tau)$, which ideally should be set to 1 if agent i receives precisely the items in C, and 0 otherwise. Furthermore, $C \in \mathscr{C}(i,\tau)$ contains two sets of constraints: The first set of constraints states that every agent should receive at least one configuration of value at least τ for him; and the second one states that every item should be assigned to at most one agent. C-LP(τ) is described by the following system of linear inequalities:

$$\begin{aligned}
\sum_{C \in \mathscr{C}(i,\tau)} x_{iC} &\geq 1 && \forall i \in \mathscr{P} \\
\sum_{i \in \mathscr{P}} \sum_{C \in \mathscr{C}(i,\tau): j \in C} x_{iC} &\leq 1 && \forall j \in \mathscr{R} \\
0 \leq x_{iC} &\leq 1 && \forall i \in \mathscr{P}, C \in \mathscr{C}(i,\tau)
\end{aligned}$$

C-LP(τ) contains an exponential number of variables; however, it is known that we can approximate the LP to any constant factor in polynomial time(see Appendix, [6]); specifically, for any $0 < \epsilon < 1$, if C-LP(τ) is feasible, then we can find a solution to C-LP($\tau(1-\epsilon)$) in time which is polynomial in $|\mathscr{R}|^{O(1/\epsilon)}$ and $|\mathscr{P}|$.

2.2 Configuration-LP for Min-Sum of Weighted Completion Times Scheduling

Next, we introduce the configuration-LP for min-sum of weighted completion times scheduling with uniform Smith ratios. Since in our special case we have that $p_{ij} \in \{w_j, \infty\}$ for all $i \in \mathscr{M}$ and $j \in \mathscr{J}$, we let $p_j = w_j$. We further let $\mathscr{J}_i = \{j \in \mathscr{J} : p_{ij} = p_j\}$ denote the set of jobs that can

be assigned to machine $i \in \mathcal{M}$.

Consider an optimal schedule of the considered instance. Observe that the schedule partitions the jobs into $|\mathcal{M}|$ disjoint sets $\mathcal{J} = C_1 \cup C_2 \cup \cdots \cup C_{|\mathcal{M}|}$, where the jobs in C_i are scheduled on machine i (so $C_i \subseteq \mathcal{J}_i$). As we will see in Chapter 4, the optimal cost of scheduling the set of jobs C_i on machine i can be written as

$$\text{cost}(C_i) = \sum_{j \in C_i} p_j^2 + \sum_{j \neq j' \in C_i} \frac{p_j p_{j'}}{2}.$$

The total cost of the considered schedule is $\sum_{i \in \mathcal{M}} \text{cost}(C_i)$.

The configuration-LP models, for each machine, the decision of which configuration (set of jobs) this machine should process. Formally, we have a variable y_{iC} for each machine $i \in \mathcal{M}$ and each configuration $C \subseteq \mathcal{J}_i$ of jobs. The intended meaning of y_{iC} is that it takes value 1 if C is the set of jobs that machine i processes, otherwise it takes value 0. The constraints of a solution are that each machine should process at most one configuration and that each job should be processed precisely once. The configuration-LP can be compactly stated as follows:

$$\begin{aligned}
\min \quad & \sum_{i \in \mathcal{M}} \sum_{C \subseteq \mathcal{J}_i} y_{iC} \, \text{cost}(C) \\
\text{s.t.} \quad & \sum_{C \subseteq \mathcal{J}_i} y_{iC} \leq 1 && \forall i \in \mathcal{M}, \\
& \sum_{i \in \mathcal{M}} \sum_{C \subseteq \mathcal{J}_i : j \in C} y_{iC} = 1 && \forall j \in \mathcal{J}, \\
& y_{iC} \geq 0 && \forall i \in \mathcal{M}, \, C \subseteq \mathcal{J}_i.
\end{aligned}$$

This linear program has an exponential number of variables and is therefore non-trivial to solve. However, Sviridenko and Wiese [45] show that, for any $\varepsilon > 0$, there exists a polynomial-time algorithm that gives a feasible solution to the relaxation whose cost is at most a factor $(1+\varepsilon)$ more than the optimum. Hence, the configuration-LP becomes a powerful tool that we use to design a good approximation algorithm for our problem.

3 Restricted Max-Min Fair Allocation

The results of this chapter are based on a joint work with Chidambaram Annamalai and Ola Svensson; this work was published in SODA 2015 [1].

3.1 Introduction

In this chapter, we study the max-min fair allocation problem. Recall that this is the problem of finding an assignment of items to agents, which maximizes the max-min fairness criterion. Formally, we are given a set $\mathscr{R}$ of n indivisible items, a set $\mathscr{P}$ of m agents, and a valuation v_{ij} for each agent i and item j, which denotes how much i cares for receiving j. The goal is to find an assignment of these items to the agents, such that the minimum total value any agent receives is maximized; in other words, we want to find disjoint sets $S_i \subseteq \mathscr{R}$ for each agent i, such that we maximize

$$\min_{i \in \mathscr{P}} \sum_{j \in S_i} v_{ij}.$$

The problem has also been called the Santa Claus problem, as it can be imagined that the agents are children, the items are Christmas presents, and we are Santa Claus who wants every child to receive at least a few presents. In this thesis, we are particularly interested in the variant of the problem called the restricted max-min fair allocation problem. In this special case, every item is interesting to only a subset of the agents and is valued the same by all these agents; formally, we have that $v_{ij} \in \{0, v_j\}$ for all $j \in \mathscr{R}$ and all $i \in \mathscr{P}$.

Using techniques that exploit the structural similarities between the max-min fair allocation problem, and the min-max scheduling problem, Bezáková and Dani [7] show that we can round the corresponding assignment-LP for the problem, to get an assignment of value $\text{OPT} - v_{\max}$, where OPT is the LP value and $v_{\max}$ is the maximum value of any item for any agent. However, since $v_{\max}$ can be arbitrarily close to OPT, this approach does not guarantee any approximation ratio; in fact, the integrality gap of the assignment-LP for this problem can be shown to be arbitrarily bad.

In view of this fact, Bansal and Sviridenko [6] propose and study the configuration-LP for the problem. Their first negative result states that the integrality gap of the configuration-LP is $O(\frac{1}{\sqrt{m}})$ in general. On the positive side, Asadpour and Saberi [4] and Saha and Srinivasan [35] designed approximation algorithms that almost match this bound, providing ratios $\Omega(\frac{1}{\sqrt{m}\log^3 m})$ and $\Omega(\frac{\log\log m}{\sqrt{m\log m}})$ respectively.

Bansal and Sviridenko show that for the restricted max-min fair allocation problem, the integrality gap improves to $\Omega(\frac{\log\log\log m}{\log\log m})$. Even more interesting is that, Bansal and Sviridenko show that a solution to a certain combinatorial problem would imply a constant integrality gap. Feige [14] proved this was possible, by providing a proof that included multiple applications of the Lovász Local Lemma, which meant that a polynomial-time approximation algorithm was not immediately provided. To remedy this, Hauepler et al. [19] gave a constructive proof of Feige's result, which provided the first polynomial time approximation algorithm with a constant approximation ratio for the problem. However, one issue remained: Due to the nature of the result, the resulting approximation guarantee is fairly large, and though it provides proof of concept of the fact that there should be an algorithm with a good approximation guarantee, the gap between the known upper and lower bounds on the approximability of the problem remained wide (notice that it is **NP**-hard to approximate the problem within a factor better than $\frac{1}{2}$ [7]).

Although the above results summarize what was known with respect to approximating the restricted max-min allocation problem, they do not sum up the best bound we have on the integrality gap of the configuration-LP for the problem. In fact, Asadpour et al. [2] provide an algorithmic proof that the integrality gap is at least $1/4$, albeit with an exponential running time. Their approach displays similarities with the approach taken by Haxell [21] to provide sufficient conditions for the existence of matchings on bipartite hypergraphs. These techniques were extended by Polacek and Svensson [32] to provide a quasi-polynomial time approximation algorithm, with an approximation ratio of $\frac{1}{4+\epsilon}$, for any $\epsilon > 0$. Although in the following section we will revisit the techniques used by Asadpour et al. and Polacek and Svensson, at this point it is worth noting that the latter algorithm is combinatorial and only uses the configuration-LP as a lower bound in the analysis.

Our Contributions The problem we approximate generalizes the maximum matching problem on bipartite graphs: it is a special case of the maximum matching problem on bipartite hypergraphs. Therefore, it should come as no surprise that the techniques developed by Asadpour et al. and Polacek and Svensson are in some sense an extension of the alternating-path technique for extending a partial bipartite matching. By constructing an *alternating tree*, whose levels alternate between containing agents and sets of items, their algorithms iteratively increase the number of agents that are matched to a set that contains items that have a significant total value. However, in both cases the running time of the alternating-tree algorithm was super-polynomial. Designing a combinatorial polynomial-time algorithm, which leads to a good approximation guarantee and is based on the idea of using alternating

trees, is one of the main contributions of this thesis:

Theorem 3.1.1. *For every $\epsilon > 0$, there exists a combinatorial $\frac{1}{(6+2\sqrt{10}+\epsilon)}$-approximation algorithm for the restricted max-min fair allocation problem that runs in time $s^{O(1/\epsilon^2 \log(1/\epsilon))}$, where s is the size of the input instance.*

Chapter Overview Throughout the following sections, we will prove the main result of this chapter, Theorem 3.1.1. In order to do so, we will first describe and analyze an $\frac{1}{4}$-approximation algorithm in Section 3.2, that was introduced by Asadpour et al. [2]. Even though this algorithm will run in exponential time, it will serve to display the main structure of an alternating tree algorithm, as well as the obstacles we will have to overcome to achieve a polynomial running time. Then, we will design and analyze an $\frac{1}{36}$-approximation algorithm that runs in polynomial time, in Section 3.3. Although this algorithm will use the ideas we sketched above to achieve polynomial running time, it will require solving the configuration-LP for the restricted max-min fair allocation problem, in order to facilitate the design of an algorithm that is as simple as possible, at the extra cost of an inferior approximation guarantee. Finally, in Section 3.4, we design and analyze the purely combinatorial algorithm promised by Theorem 3.1.1.

3.2 A Simple Alternating-Tree Algorithm

In order to expose some basic concepts behind the design and analysis of alternating-tree algorithms, we start by describing a basic $\frac{1}{4}$-approximation algorithm for the restricted max-min fair allocation problem. However, it runs in exponential time. Similar to the algorithm we will describe in Section 3.4, this algorithm will be combinatorial, hence we will employ the configuration-LP only as part of the analysis.

The main result of this section is the following:

Theorem 3.2.1. *There exists an algorithm that, given an instance of the restricted max-min fair allocation problem, and given $\tau > 0$, either returns a solution of value at least $\tau/4$, or asserts that C-LP(τ) is infeasible; furthermore, the running time of this algorithm is $O(n^2 m 2^m)$.*

By applying binary search on the possible values of τ, it follows that the above theorem directly implies an $\frac{1}{4}$-approximation algorithm that runs in $O(n^2 2^m m \log \mathrm{OPT}) = O(s^4 2^s)$ time, where s is the size of the description of the input instance.

3.2.1 Notation

We begin by introducing some basic notation. To begin with, we make the distinction between *fat* and *thin* items; specifically, we partition the set of items $\mathscr{R}$ into the set of fat items $\mathscr{R}_f = \{j \in \mathscr{R} : v_j \geq \tau/4\}$ and the set of thin items $\mathscr{R}_t = \mathscr{R} \setminus \mathscr{R}_f$. Remember that we say agent i is interested in item j if $v_{ij} = v_j > 0$. Clearly, if we want to find a solution whose value is at

least $\tau/4$, then for any agent i, it suffices to assign to i a single fat item, among those that i is interested in; otherwise, we should assign to him multiple thin items that he is interested in, of total value at least $\tau/4$.

As we have already mentioned in the previous section, we will design an alternating-tree algorithm; the basic structure of this algorithm will be called an *edge*:

Definition 3.2.1. *We call a pair (p,R), where $p \in \mathscr{P}$ and $R \subseteq \mathscr{R}$, an edge, if R is a subset of items of total value at least $\tau/4$ that p is interested in.*

Similar to how we distinguished between fat and thin items, we define *fat* and *thin edges*:

Definition 3.2.2. *We call an edge consisting of a single fat item a fat edge.*

Definition 3.2.3. *We call an inclusion-wise minimal edge (i.e., an edge for which removing any item would cause the total value of items to drop below $\tau/4$) consisting of multiple thin items a thin edge.*

In fact, from now on we will only consider inclusion-wise minimal edges.

Finally, we define *partial matchings*:

Definition 3.2.4. *We call a set M of pairwise disjoint fat and thin edges (i.e., edges that share no agents or items) a (partial) matching. Furthermore, we will say that the agents that appear in the edges of M are matched by M.*

Observe that using the above notation, the purpose of Theorem 3.2.1 is to design an algorithm that returns a matching that matches all agents. For any set of edges E, let $\mathscr{P}(E)$ be the agents that appear in some edge in E. The following lemma directly implies Theorem 3.2.1:

Lemma 3.2.1. *There exists an algorithm that, given an instance of the restricted max-min fair allocation problem, $\tau > 0$, a partial matching M and an agent $p_0 \notin \mathscr{P}(M)$, either returns a partial matching M' such that $\mathscr{P}(M') = \mathscr{P}(M) \cup \{p_0\}$, or asserts that C-LP($\tau$) is infeasible. Furthermore, the running time of this algorithm is $O(nm2^m)$.*

The algorithm behind Theorem 3.2.1 simply entails running the algorithm of Lemma 3.2.1 until all agents are matched, or until we terminate with failure, which would signify that our guessed optimum τ was incorrect.

The rest of this section is dedicated to proving Lemma 3.2.1.

3.2.2 Algorithm Overview

Before proceeding to the formal definition and analysis the promised algorithm, we will illustrate some of its basic concepts through an example, that appears in Figure 3.1.

To begin with, we are given a partial matching M, and we are asked to extend this to match agent p_0. In order to do so, we check whether there exist items of total value $\tau/4$, which do not appear in any edge in M, that p_0 is interested in. If this is the case, then we include the induced edge in our matching. Otherwise, we find any minimal subset of items, of value at least $\tau/4$, which p_0 is interested in, and try to add the corresponding edge into M; this is the gray edge in Figure 3.1(a). However, some of the items in this gray edge will appear in some of the edges of M; these are the white edges in Figure 3.1(a), and they block (the concept of a blocking edge will be formally defined later on) the insertion of the gray edge to M.

In order to add the gray edge into M, we first need to free up the items it shares with the edges that block it. In turn, in order to free up these items, we will have to match the agents that appear in these blocking edges by using edges whose items are disjoint from the items in M. We show this situation in Figure 3.1(b); we found a fat item that does not appear in any edge in M, which we will use to satisfy the agent that appears in the first blocking edge. Therefore, we will include this fat edge into M.

At this point, we are left with the edges that appear in Figure 3.1(c). Next, we need to find an alternative way of satisfying the agent that appears in the remaining blocking edge. We take up the same approach we did in order to satisfy agent p_0: we find an edge we want to add into M, and we consider the edges blocking its inclusion. This leaves us with the edges that appear in Figure 3.1(d). By iterating this process, we will try to find new edges that satisfy the agents that appear in the new blocking edges, and we continue until we are able to free up all the items in the original gray edge (i.e., the one containing agent p_0).

By observing the emerging structure, we can see where the term *alternating-tree algorithm* stems from: the edges we want to include in M, and the edges that block them, form a tree of hyperedges, where a gray edge is only connected to white edges, and vice versa.

Next, we proceed to formalize the process we sketched above.

3.2.3 A Basic Alternating-Tree Algorithm

In this section, we will design a procedure that starts from a partial matching M, and extends it to include agent p_0. Applying this process iteratively, we will be able to retrieve a matching that matches all the agents.

State of the Algorithm At any point during its execution, the state of the algorithm can be described by a tuple (M, p_0, A, B): M is our tentative partial matching, p_0 is the agent we want to include in M, A is the set of edges we want to include in M, and B are the edges of M that block the inclusion of the edges of A to M (i.e., they are the edges of M that intersect the edges of A).

Before moving on to the formal description of the promised algorithm, we need to formally

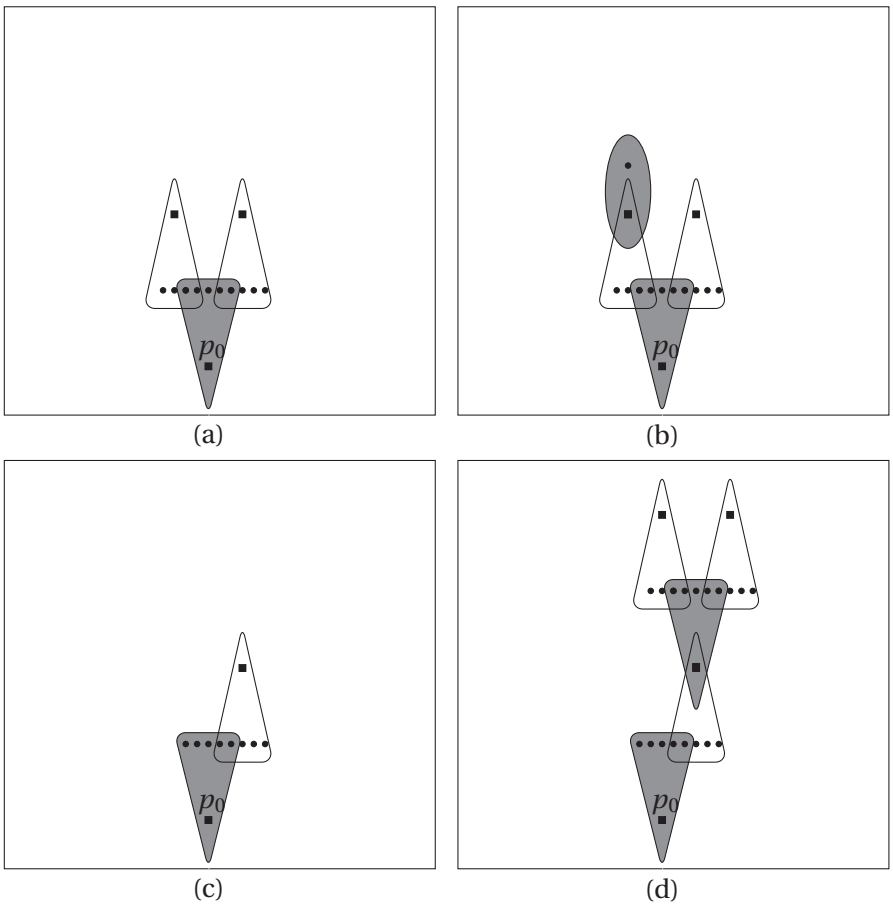

Figure 3.1: An example execution of our basic algorithm. In this figure, boxes correspond to players, and circles correspond to resources.

define the concepts of *addable*, *immediately addable* and *blocking* edges:

Definition 3.2.5. *Given the state* (M, p_0, A, B) *of our algorithm:*

- *We call an edge* $e = (p, R)$ addable, *if* $p \in \mathscr{P}(B) \cup \{p_0\}$ *and there exists no edge* $e' \in A \cup B$ *such that* e *and* e' *have an item in common.*
- *We call an edge* $e = (p, R)$ immediately addable, *if* $p \in \mathscr{P}(B) \cup \{p_0\}$ *and there exists no edge* $e' \in A \cup B \cup M \setminus \{e\}$ *such that* e *and* e' *have an item in common.*
- *We say that an edge* $e \in M$ *is* blocking *for an edge* e' *if they share an item.*

We are now ready to describe the promised algorithm, that was introduced by Asadpour et al. [2]:

Initialization: Initially, $A = B = \emptyset$. If there exists an immediately addable edge e that contains p_0, then

$M \leftarrow M \cup \{e\}$, and we terminate with success; otherwise, we choose an addable edge e which contains p_0, insert e into A and insert all edges that block e into B.

Iterative step: The iterative step consists of two phases:

Build phase: If there exists no agent $p \in \mathscr{P}(B) \cup \{p_0\}$ that belongs to some addable edge, terminate with failure; otherwise, choose an addable edge e that contains some $p \in \mathscr{P}(B) \cup \{p_0\}$. Then, insert e into A and insert all blocking edges of e into B.

Collapse phase: While there exists an immediately addable edge e in A, if $\mathscr{P}(\{e\}) = \{p_0\}$, insert e into M and terminate with success. Else, consider the edge $e' \in B$ such that $\mathscr{P}(\{e\}) = \mathscr{P}(\{e'\})$, and set $M \leftarrow M \setminus \{e'\} \cup \{e\}$ and $B \leftarrow B \setminus \{e'\}$.

Iterate until there are no addable edges and all edges in A are blocked by at least one edge in B, in which case we terminate with failure.

Next, let us explain the main steps of the algorithm. During the initialization phase, we set initial values for A and B, and then we check whether there exists an immediately addable edge for p_0: If there is, then we can simply insert this edge into our partial matching M, which would imply we extended our matching to include p_0. If there exists no immediately addable edge that contains p_0, we choose an addable edge e that contains p_0, and insert it into A: this is an edge we want to include in M. However, in M there already exist edges that share items with e, i.e., blocking edges for e; these edges we insert into B. Observe that, as we insert addable edges into A, the items of any two edges in A will always be pairwise disjoint.

Next, the iterative step of our algorithm consists of two distinct phases: the build and collapse phases. During the build phase, we choose an addable edge e that we insert into A, and we insert all of its blocking edges into B. During the collapse phase, we find an edge $e \in A$ that is immediately addable: this is an edge we can insert into our partial matching M. To do so, we remove the edge in $M \cap B$ that contains the agent of e, and insert e into M. Notice that the agents that are matched by M do not change. Furthermore, observe that as any edge e that is introduced into M was previously immediately addable, the items of e are disjoint from items in any edge in $A \cup M \setminus \{e\}$. As the collapse phase is the only part of the algorithm that modifies M, when the algorithm terminates the agents that are matched by M are those that were initially matched, plus agent p_0 if the algorithm terminated with success.

Let us now proceed with the analysis of the above algorithm. We need to ensure two facts: (1) that the running time is $O(nm2^m)$ (as every execution of the iterative step takes $O(nm)$ time, it will suffice to prove that the iterative step is executed at most 2^m times), and (2) that termination with failure implies that C-LP(τ) is infeasible.

3.2.4 Running-Time Analysis

Throughout the execution of the algorithm, let us denote $A = \{a_1 ... a_{|A|}\}$, where $i < j$ implies that a_i was inserted in A before a_j. Furthermore, at any time during the execution of the

algorithm, let m_i be the number of edges that block a_i. We will use $\{m_i : i \in [|A|]\}$ to mark the progress our algorithm has made in the following way: at any time, we define the following m-dimensional signature vector

$$M = (m_1, ..., m_{|A|}, m+1..., m+1)$$

In order to prove that the algorithm terminates in $O(nm2^m)$ time, we need to prove the following two facts:

Fact 3.2.1. *Throughout the execution of the algorithm, every possible configuration of the signature vector appears at most once.*

Proof. In order to prove this, it suffices to show that every iterative step decreases the lexicographic order of the signature vector. On the one hand, let M_1 (M_2) be the signature vector before (after) one execution of the build phase of any iterative step, and let A_1 (A_2) be the set A of edges kept by the algorithm before (after) that execution of the build phase of the iterative step. Furthermore, let $M(i)$ denote the i-th coordinate of a signature vector M. If during the build phase, we introduced an addable edge to A_1, then M_1 and M_2 are identical up to position $|A_1|$, and $M_1(|A_1|+1) > M_2(|A_1|+1)$, as the addable edge that was inserted cannot have more than m blocking edges.

On the other hand, let M_1 (M_2) be the signature vector before (after) one execution of the collapse phase of any iterative step, and let A_1 (A_2) be the set A of edges kept by the algorithm before (after) that execution of the collapse phase of the iterative step. If during the collapse phase an immediately addable edge e was chosen, then either the algorithm terminates (if the immediately addable edge contained p_0), or there exists some e' such that e' belonged to B at the beginning of the iterative step, but was replaced by e in M after the execution of the iterative step. Consequently, if e' was blocking edge a_i, then $M_1(i) > M_2(i)$ and for all $j < i$ $M_1(j) = M_2(j)$ (observe we only removed one blocking edge during this collapse phase, hence the number of blocking edges of a single edge in A decreased). In both cases, the lexicographic order of M_2 is less than that of M_1. □

Fact 3.2.2. *There exist at most 2^m possible configurations of the signature vector.*

Proof. The key observation behind this fact is that $\sum_i m_i \leq m$ at any time during the execution of the algorithm, because there can be no more than m blocking edges. Then, the claim follows by observing that we can put the set of all possible configurations of the signature vector in a 1-1 correspondence with $\{0,1\}^m$; specifically, we correspond $M = (m_1, ..., m_{|A|}, m+1..., m+1)$ with the 0-1 vector

$$0^{m_1-1}10^{m_2-1}1...0^{m_{|A|}-1}10^{m-\sum_i m_i}$$

Since the above vector contains $m_{|A|}$ ones and $\sum_{i \leq m_{|A|}} (m_i - 1) + m - \sum_{i \leq m_{|A|}} m_i = m - m_{|A|}$ zeros, its total length is m; therefore, there are at most 2^m such vectors, hence, 2^m corresponding

signature vectors. □

3.2.5 Correctness Analysis

Finally, in order to prove that the algorithm satisfies the requirements of Lemma 3.2.1, we will prove the following lemma:

Lemma 3.2.2. *Whenever the algorithm terminates with failure, the C-LP(τ) is infeasible.*

Proof. Let us assume the algorithm terminates with failure, and let (M, p_0, A, B) be the state of our algorithm at that point. In order to prove that the C-LP(τ) is infeasible, we will use A and B to construct a feasible solution for the dual LP of the C-LP(τ), whose objective value is strictly positive. From the definition of the dual LP, it is clear that any solution with a strictly positive objective value t, can be scaled by any $c \in \mathbb{R}^+$ to obtain a solution of value ct; in turn, this will imply that the dual LP is unbounded, which in turn implies that the primal LP is infeasible.

First of all, let us write down the dual LP of the C-LP(τ):

$$\begin{array}{rlll} \max & \sum_{i \in \mathscr{P}} y_i - \sum_{j \in \mathscr{R}} z_j & & \\ \text{subject to} & y_i \leq \sum_{j \in C} z_j & & \forall\, i \in \mathscr{P}, C \in \mathscr{C}(i,\tau) \\ & y_i \geq 0 & & \forall\, i \in \mathscr{P} \\ & z_j \geq 0 & & \forall\, j \in \mathscr{R} \end{array}$$

We define the following solution for the dual LP:

$$y_i = \begin{cases} \frac{3}{4}, \text{ if } i \text{ appears in some edge in } A \cup B \\ 0, \text{ otherwise} \end{cases}$$

$$z_j = \begin{cases} \frac{3}{4}, \text{ if } j \text{ is a fat item that appears in some edge in } A \cup B \\ \frac{v_j}{\tau}, \text{ if } j \text{ is a thin item that appears in some edge in } A \cup B \\ 0, \text{ otherwise} \end{cases}$$

Let us check that the solution (y, z) we described is feasible; clearly, we only have to focus on agents i such that $y_i = \frac{3}{4}$. Now, if we consider the constraint corresponding to any i such that $y_i = \frac{3}{4}$, and any $C \in \mathscr{C}(i,\tau)$ such that C contains a fat item, then from the definition of (y, z) it immediately follows that the constraint will be satisfied, as all fat items that can be assigned to i appear in $A \cup B$ (if such an item did not appear in $A \cup B$, the assumption that the algorithm terminated with failure would be wrong, since that item would constitute a valid fat addable edge), whereas if we look at any constraint corresponding to some i and some

$C \in \mathscr{C}(i,\tau)$ consisting only of thin items, then $\sum_{j\in C} z_j < y_i$ would imply that

$$\sum_{j\in C'} \frac{v_j}{\tau} < \frac{3}{4}$$

where $C' \subseteq C$ are the items of C that appear in an edge in $A \cup B$. Then, this would imply

$$\sum_{j\in C\setminus C'} v_j > \frac{\tau}{4}$$

which is a contradiction. To see this, notice that $(i, C \setminus C')$ constitutes a valid addable edge we did not insert into A, thus the algorithm could not have terminated with failure without including it.

Finally, let us prove that the objective value achieved by (y,z) is strictly positive: Let A_f (A_t) be the set of fat (thin) edges in A, and let B_f (B_t) be the set of fat (thin) blocking edges in B. First, we observe that $\sum_{i\in\mathscr{P}} y_i = \frac{3}{4}(|B|+1)$, because p_0 is the only agent that appears in an edge in $A \cup B$, that does not appear in a blocking edge, and as there is no agent that appears in multiple blocking edges: To see this last fact, observe that blocking edges belong to M, at the beginning of our algorithm M is a valid partial matching, and according to the collapse phase of our algorithm, any time we insert an edge e in M, we remove the edge that contains $\mathscr{P}(e)$ from M. Furthermore, we observe that any thin edge in A contains items of total value at most $\frac{\tau}{2}$. To see this, note that if it were not true, then the edge would not be inclusion-wise minimal, as it would contain items of value greater than $\tau/2$, each of which has value at most $\tau/4$. Hence, removing a single item leaves items of total value greater than $\tau/4$, which is a contradiction.

Next, observe that any thin blocking edge contains items that do not appear in an edge in A of total value at most $\frac{\tau}{4}$ (again, the contrary would imply that blocking edge is not minimal inclusion-wise). This, combined with the fact that, for thin items, $z_j = \frac{v_j}{\tau}$, this implies that the sum of z-values of the items contained in any thin edge in A is at most $\frac{1}{2}$, and that the sum of z-values of all the items contained in any thin blocking edge that do not appear in another edge in A is at most $\frac{1}{4}$. Finally, we observe that there are exactly $|B_f|$ fat items in $A \cup B$ (because if there were an addable fat edge that was not blocked, the termination with failure assumption would be contradicted), and that $|A_t| \leq |B_t|$ (because, for the algorithm to terminate with failure, every thin edge must be blocked by some other thin edge). Therefore, we have

$$\begin{aligned}\sum_{i\in\mathscr{P}} y_i - \sum_{j\in\mathscr{R}} z_j \geq \\ \frac{3}{4}(|B_f|+|B_t|+1) - \frac{3}{4}|B_f| - \frac{1}{2}|A_t| - \frac{1}{4}|B_t| \geq \\ \frac{3}{4}(|B_t|+1) - \frac{3}{4}|B_t| = \frac{3}{4}.\end{aligned}$$

As the objective value of the dual LP for solution (y, z) is strictly positive, the proof of Lemma 3.2.1 (and therefore of Theorem 3.2.1) is concluded. □

3.2.6 Novel Concepts for Improved Running-Time

Next, we will discuss some of the new concepts that are introduced in this thesis, in order to design an approximation algorithm for the restricted max-min fair allocation problem that runs in polynomial time. To begin with, we should note that the algorithms we will describe in the following section will again use the concept of constructing a hypergraph of addable and blocking edges, and then use a signature vector argument in order to bound the running time.

The main question we will face, is how to design such an algorithm, in a way that the number of possible signature vectors is small. In order to answer this question, the first new idea we will introduce is that of *greedy agents*: These will be agents, whose addable edges contain items of total value significantly more than ρOPT, where ρ is the approximation ratio we are aiming for. The idea behind having such agents in our alternating tree is straightforward: As these agents claim more items than they would be satisfied with, the rate of growth of the alternating tree is boosted. To be more precise, the number of blocking edges that appear on each level (i.e., the number of blocking edges at a specific distance from the root) of our alternating tree will increase exponentially, which implies that the number of possible configurations of the signature vector we might go through decreases drastically.

The second new idea we introduce, which will complement the existence of greedy agents, is that of *lazy updates*: As we explained before, one of the main points of the alternating-tree framework is that, whenever we find an addable edge e for an agent q that appears in a blocking edge e', such that e contains no items that already appear in our partial matching, we are able to substitute e' with e, therefore freeing up the items in e' and making progress. The main idea behind performing lazy updates is that, instead of performing such an update whenever we find an appropriate addable edge, we will postpone it until we are able to ensure that we can perform such an update on a significant amount of the agents that simultaneously appear in blocking edges. Consequently, we will go through fewer configurations of the signature vector, because instead of making many small updates, we will make a few large ones. Thus, combining this idea with the greedy agents idea, we will be able to ensure we only go through a polynomial number of configurations of the signature vector.

3.3 Utilizing Greediness and Laziness

As we already mentioned in the introduction of this chapter, in order to expose the main ideas behind our approach, in this section we will provide a polynomial-time approximation algorithm, with an inferior approximation guarantee that will work on input instances that have undergone some preprocessing (we will call these preprocessed instances *clustered*). Formally, the main result of this section is the following:

Theorem 3.3.1. *There exists an $\frac{1}{36}$-approximation algorithm for the restricted max-min fair allocation problem.*

Naturally, we will start by describing what these clustered instances look like.

Clustered Instances The preprocessing step we will use was introduced by Bansal and Sviridenko [6], and its functionality can be formally described as follows (we remark that, since we are aiming for an $\frac{1}{36}$-approximation guarantee, in this section fat items will be those whose value is at least $\tau/36$, where τ is the guessed optimum):

Theorem 3.3.2 ([6])**.** *If C-LP(τ) is feasible, we can partition the set of agents $\mathscr{P}$ into m clusters $N_1, \ldots, N_m$ in polynomial time such that*

1. *each cluster N_k is associated with a distinct subset of $|N_k|-1$ fat items from $\mathscr{R}_f$ such that they can be assigned to any subset of $|N_k|-1$ agents in N_k, and*
2. *there is a feasible solution x to C-LP(τ) such that $\sum_{i\in N_k}\sum_{C\in\mathscr{C}_t(i,\tau)} x_{iC} = 1/2$ for each cluster N_k, where $\mathscr{C}_t(i,\tau)$ denotes the set of configurations for agent i comprising only thin items.*

As we can choose the agent that does not receive a fat item for each cluster independently, in order to design an $\frac{1}{36}$-approximate allocation, it suffices to choose one agent from each cluster to receive a thin edge of total value at least $\frac{\tau}{36}$. This is precisely the advantage of this preprocessing: We can focus on assigning one thin edge for one agent for each cluster, therefore we are able to design an algorithm that will be completely oblivious to fat items/edges. Removing the clustered instance assumption, thus improving our approximation guarantee, will be our focus in Section 3.4.

In this context, we need to redefine thin items, thin edges and partial matchings:

Definition 3.3.1. *A* thin item *is an item whose value is at most $\tau/36$; furthermore, we denote the thin items by $\mathscr{R}_t$, and the fat items $\mathscr{R}\setminus\mathscr{R}_t$ by $\mathscr{R}_f$.*

Definition 3.3.2. *A* thin edge *is a pair (p,R), where p is an agent and R is an inclusion-wise minimal set of thin items p is interested in, of total value at least $\tau/36$.*

Definition 3.3.3. *A* partial matching *M is a set of thin edges of total value at least $\tau/36$, such that no two edges in M contain the same item, and no two edges in M contain agents from the same cluster.*

We will say that M matches agent p (cluster N_k) if M contains a thin edge that includes p (an agent from N_k respectively).

Using arguments similar to those used in Section 3.2, we can see that in order to prove Theorem 3.3.2, it suffices to prove the following lemma:

Lemma 3.3.1. *There exists a polynomial time algorithm that, given a partial matching M, and given a cluster N_0 not matched by M, extends M to match N_0 as well.*

In the following sections, we show how to prove the above lemma. Before we proceed, however, with describing and analyzing the promised algorithm, let us conduct an overview of how the algorithm roughly works, through an example.

3.3.1 Algorithm Overview

For a particular guess τ, let us assume that C-LP(τ) is feasible. Our goal now is to satisfy each cluster, i.e., allocate a disjoint collection of items of value at least $\tau/36$ for exactly one agent in each cluster. To this end, we design an iterative procedure that we will apply in order to find such an $1/36$-approximate allocation. The input to this procedure will be a partial matching and an unmatched cluster N_0. In order to satisfy the input cluster as well, our algorithm will extend the partial matching. Thus, applying this procedure iteratively will satisfy all the clusters.

Here, we illustrate some key aspects of this procedure through an example that appears in Figure 3.2. Given a partial matching, we want to extend this to match agent p from cluster N_0. If there are free items (i.e., not already appearing in our partial matching) of total value $\tau/36$ for p, then we simply satisfy p by assigning those items to him. Otherwise, we find a set of items whose value for p is at least $2\tau/5$; these items constitute an edge e_p we would want to include in our partial allocation in order to satisfy p. However, we cannot include this edge immediately because in our partial allocation there already exist edges that share items with e_p: In other words, such edges *block* the inclusion of e_p into our partial allocation. In Figure 3.2(a), e_p is the gray edge, and its blocking edges are the white ones.

At this point, we should note that the size of e_p is considerably larger than our goal of $\tau/36$; this is by design and due to our *greedy* strategy. By considering edges whose sizes exceed our goal, we are able to increase the rate at which blocking edges are considered by our algorithm; indeed, in Figure 3.2(a), a single greedily constructed edge (e_p) introduced three blocking edges. Ultimately, this enables us to bound the running time of our algorithm.

Now, as our goal is to include e_p in our partial matching, we need to free up some of the items of e_p by finding an alternative way of satisfying the agents included in the blocking edges of e_p. The steps we take towards this end appear in Figure 3.2(b): for each agent in the blocking edges of e_p, we find in the same cluster a new edge for some agent, who we want to include into our partial matching. In this figure, agents that belong to the same cluster are enclosed in a dashed circle. However, these new gray edges might also be blocked by existing edges in our partial matching. Therefore this step introduces a second *layer* of edges we want to include in our allocation, and their corresponding blocking edges; in the example, these layers are separated by dashed lines.

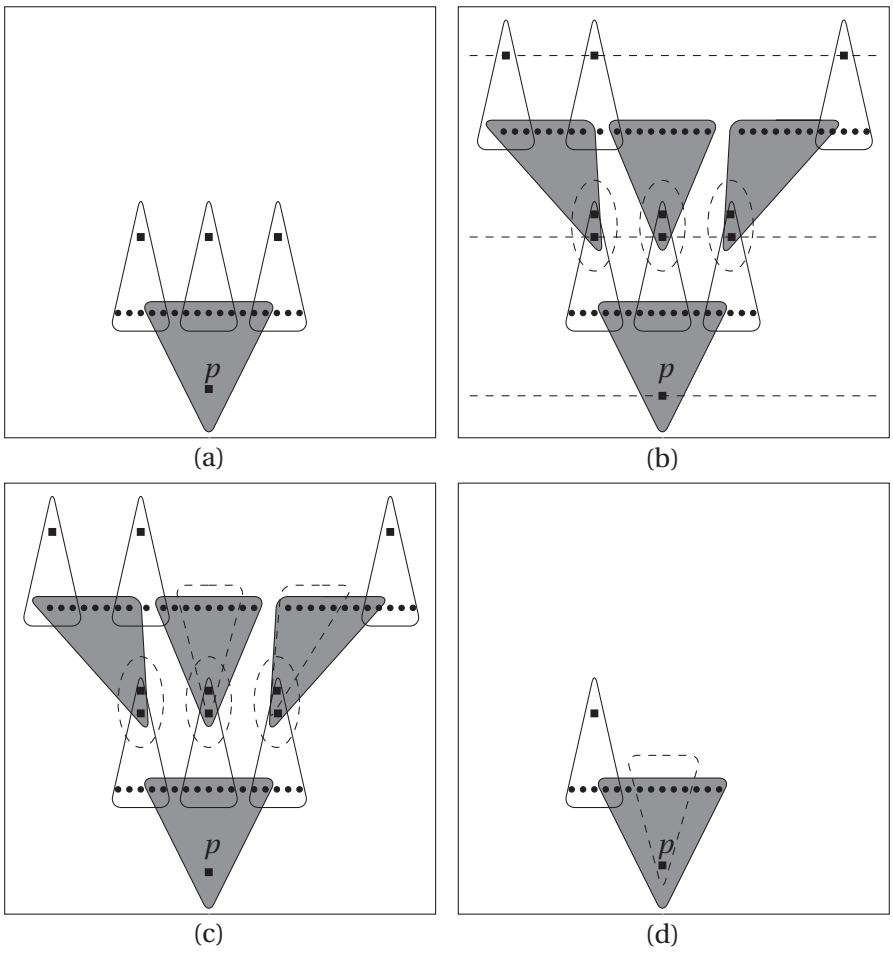

Figure 3.2: An example execution of our algorithm for clustered instances. In this figure, boxes correspond to players, and circles correspond to resources.

Next, we observe that two of the three gray edges in the second layer have many items that do not appear in any blocking edge. In this case, as one can see in Figure 3.2(c), we select a subset of free items from each edge of size at least $\tau/36$ (drawn with dashed lines), and swap these edges for the existing white edges in our partial matching. We call this operation a *collapse* of the second layer, after which we are left with e_p and a single blocking edge in the first layer. The way we decide when to collapse a layer, is dictated by our strategy of *lazy updates*: As in Figure 3.2(c), we will only collapse a layer if that would mean that a large fraction of the previous layer's blocking edges will be removed.

Finally, in Figure 3.2(d), a significant amount of items of e_p has been freed up. Then, we choose a subset of these items (again, drawn with a dashed line) and allocate them to p. At this point, we have satisfied p and succeeded in extending our partial matching to satisfy one more agent and one more cluster.

Next, we proceed with formally defining and analyzing the local-search algorithm we sketched

above.

3.3.2 An Algorithm for Clustered Instances

The next step we take is to describe and analyze the algorithm for clustered instances. This algorithm will be behind the proof of Theorem 3.3.1. Note that as we are working on clustered instances, our algorithm will only pertain to constructing and assigning edges of thin items, which will greatly simplify the algorithm's design.

Let us now begin to introduce some of the definitions that are necessary to describe our algorithm.

Parameters Throughout the description of our algorithm, we will use the following parameters:

- $\rho = 1/\beta = 1/36$ is the approximation guarantee of our algorithm.
- $\alpha = 5/2$ is the parameter that determines the how greedy our agents are.
- $\mu = 1/500$ is the parameter that determines the laziness of our updates.

State of the Algorithm In order to describe our algorithm, we need the following definition:

Definition 3.3.4. *For any $\delta \geq 1$, we call an edge (p, R) a* δ-thin edge*, or a* δ-edge*, if R is an inclusion-wise minimal set of thin items that p is interested in, of total value at least τ/δ.*

An important property of δ-edges that we will use later on is the following:

Remark 3.3.1. *A δ-edge contains items of total value less than $\frac{\tau}{\delta} + \frac{\tau}{\beta}$, due to its minimality. Furthermore, given a thin β-edge (p, R), any strict subset of R contains items of total value at most τ/β.*

Now, we can define what a state of our algorithm is:

Definition 3.3.5. *A state of the algorithm is a tuple $(M, p_0, \ell, \{A_0, ..., A_{\ell+1}\})$, where:*

- *M is a partial matching.*
- *p_0 is the agent we want to match.*
- *ℓ is an index that keeps track of the depth of our search.*
- *A_i is a set of α-edges, with $A_0 = \emptyset$.*

Similar to the algorithm in Section 3.2, the edges in some A_i correspond to edges we want to include into our matching: Naturally, their inclusion might be blocked by edges that already exist in our partial matching. Thus, for $i > 0$, let B_i be the set of edges of M that share items with at least one edge in A_i; we conventionally set $B_0 = \{(p_0, \emptyset)\}$.

Now, notice that although ℓ denotes the depth of our search, in the state of the algorithm we define A_i for all $i \leq \ell + 1$. The reason for this disparity is that our algorithm will consist of an iterative step, a main phase of which will be to find edges we want to introduce into our matching. During this phase of each iteration, $A_{\ell+1}$ will be initially empty, and we will proceed to insert edges into it, one by one. After we are done inserting edges into $A_{\ell+1}$, we will increment ℓ and proceed to the next phase of the iterative step. Therefore, $A_{\ell+1}$ will only be used (non-empty) at the first phase of each iteration, but for the sake of formality, we include it into the definition of the state of the algorithm.

Our algorithm will try to match p_0, by inserting into M a thin β-edge that contains p_0. As our end-goal is to start from an empty matching and to iteratively apply our algorithm until all clusters are matched, and as our algorithm will only introduce β-edges into M, we will maintain the following property for M:

Remark 3.3.2. *M is a partial matching that only contains β-edges.*

Algorithm Invariants For any collection of sets $\{S_0, S_1, ..., \}$, let $S_{\leq i} = \cup_{j \leq i} S_j$. Then, as we will see later on, the state of our algorithm satisfies the following invariants at the beginning of each iterative step:

(a) For any $i \leq \ell$, there is no edge $e \in A_i$ that shares items with an edge in $A_{\leq i} \cup B_{\leq i-1}$

(b) For any $i \leq \ell$, any two edges in A_i contain agents from different clusters. Furthermore, if agent $p \in N_k$ appears in an edge in A_i, then there exists $q \in N_k$ (not necessarily different from p) that appears in an edge in B_{i-1}.

Later on, these invariants will help us show that the output of our algorithm will be indeed a partial matching.

Algorithm Description Let us introduce some final notation, that will make the description of our algorithm easier. Let $P_i = \mathscr{P}(B_i)$. Similar to Section 3.2, we define addable, immediately addable and blocking edges:

Definition 3.3.6. *Given a state of the algorithm, we call an α-edge $e = (p, R)$* addable*, if (a) there exists $q \in \mathscr{P}(B_\ell)$ in the same cluster as p, and there exists no $q \in \mathscr{P}(A_{\ell+1})$ in the same cluster as p, and if (b) R does not intersect any edge in $A_{\leq \ell+1} \cup B_{\leq \ell}$.*

Definition 3.3.7. *Given a state of the algorithm, we call an edge $e = (p, R)$* immediately addable, *if e belongs to some A_i, and if there exists $R' \subseteq R$ such that the total value of items in R' is at least τ/β, and none of the items in R' appear in any edge in M.*

Definition 3.3.8. *Given a state of the algorithm, we call an edge $e \in M$ that belongs to some B_i a* blocking *edge; similarly, given an edge e' that belongs to some A_i and an edge $e \in M$ that shares items with e', we will say that e* blocks *e'.*

One of the new features of the algorithm we describe in this section is that we iteratively construct a hypergraph of edges in layers:

Definition 3.3.9. *Given a state of the algorithm, we let $L_i = (A_i, B_i)$ denote the i-th* layer *of our algorithm.*

Definition 3.3.10. *Given a state of the algorithm, we call a layer L_i* collapsible, *if A_{i+1} contains at least $\mu|P_i|$ immediately addable edges.*

Naturally, we will call $L_{\leq \ell+1}$ the *layered hypergraph* maintained by our algorithm.

Now, we are ready to formally describe the layered hypergraph[1] algorithm for clustered instances:

Initialization: Select some agent $p_0 \in N_0$. Set $A_0 \leftarrow \emptyset$, and $\ell \leftarrow 0$. Recall that $B_0 = \{(p_0, \emptyset)\}$ by convention.

Iterative Step: The iterative step consists of two phases, that get repeated until N_0 is matched by M.

Build Phase: Set $A_{\ell+1} \leftarrow \emptyset$. While there exists an addable edge e, we add it to $A_{\ell+1}$. When there are no more addable edges, increment ℓ.

Collapse Phase: If there exists a collapsible layer, then let L_t be the earliest collapsible layer. For each edge $(p, R) \in A_{t+1}$, consider agent $q \in P_t$ such that p, q belong to the same cluster. If (p, R) is immediately addable, swap q's edge in M with (p, R'), where R' is a τ/β-minimal subset of R that shares no items with any edge in B_{t+1}.

After we process all immediately addable edges, we discard all the layers with index greater than t and set ℓ to be t. As the collapse operation could have created immediately addable edges in L_t, we repeat the collapse phase until there are no collapsible layers.

Next, let us explain the main steps of the algorithm behind Lemma 3.3.1. In the initialization phase, we choose an agent p_0 from the cluster N_0 we want to match: This will be the first agent for which we will try to find an addable edge. As we introduce no edges into our layered hypergraph in this phase, our two invariants trivially hold throughout it.

[1] In contrast to the algorithm in Section 3.2 that worked by building up an alternating tree, the algorithm we will describe builds up a layered hypergraph.

After the initialization phase, we proceed to our main iteration. Each iterative step consists of two phases: the build and collapse phases. During the build phase, the algorithm tries to find, for each cluster that contains an agent in P_ℓ, an addable α-edge for any agent in that cluster; such edges are inserted into $A_{\ell+1}$. Due to the definition of addable edges, the first invariant is upheld during this step. Furthermore, as, according to the definition of addable edges and the definition of the build phase, we only introduce into $A_{\ell+1}$ at most one edge for each cluster that contains an agent in P_ℓ, the second invariant is upheld.

During the collapse phase, we identify the lowest collapsible layer t. Then, for each immediately addable edge (p,R) in A_{t+1}, which corresponds to edge (q,S) in B_t (remember the second invariant was upheld at the beginning of the iteration of the collapse phase), we replace (q,S) with (p,R') in M, where $R' \subseteq R$ is a minimal set of items of total value at least τ/β that shares no items with any edge in B_{t+1}. Notice that every iteration in the collapse phase preserves the first invariant, as (p,R') is disjoint from any edge in $A_{\leq t+1}$ (as (p,R) initially belonged to A_{t+1} and the first invariant held at the beginning of the iteration of the collapse phase) and disjoint from any edge in M (due to the definition of R' and B_{t+1}). Furthermore, every such iteration also preserves the second invariant for all layers up to t, as A_t and B_{t-1} are not modified during any iteration. As after the last iteration of the collapse phase we discard all layers above t, the second invariant is upheld overall.

Finally, observe that the collapse phase of the algorithm is the only part that modifies M, that every edge inserted into M is disjoint from all other edges which already belong to M (due to the first invariant, the definition of immediately addable edges and the way we choose the induced edges we insert into M), and that the collapse phase does not change the set of clusters that M matches: Consequently, M always constitutes a valid partial matching that always matches the same clusters. Hence, if we ever find an immediately addable edge in A_1, (due to the second invariant) our algorithm will output a valid extension of M that also matches cluster N_0.

3.3.3 Analyzing the Algorithm for Clustered Instances

Let us now proceed with analyzing the running time and correctness of the above algorithm, i.e., let us prove Lemma 3.3.1. The first thing we want to do is to quantify how fast the layers of our algorithm grow: As we have already mentioned, our design goal was to guarantee that the number of edges of a layer increases exponentially, as the layer index increases. The following lemma quantifies this intuition:

Lemma 3.3.2. *Assuming that C-LP(τ) is feasible, at the beginning of each iterative step, $|A_{i+1}| \geq |P_{\leq i}|/5$ for each $i = 0,\ldots,\ell-1$.*

Before proving the above lemma, we state a fact we will use in the lemma's proof:

Fact 3.3.1. *Let q be an agent from some cluster N_k. If an agent q is part of some blocking edge in layer L_i, i.e., $q \in P_i$, and furthermore, there exists no edge (p,R) in A_{i+1} such that p and q*

belong to the same cluster N_k after the build phase of layer L_{i+1}, then none of the agents in N_k have a set of items of value at least τ/α that do not appear in any edge in the layered hypergraph.

This fact follows in a straightforward manner from the design of the build phase of our algorithm: More specifically, it follows because during the build phase of layer L_{i+1}, for each agent p in P_i, we find an addable edge that contains an agent in the same cluster as p, unless there exists no such edge. We proceed to prove Lemma 3.3.2:

Proof of Lemma 3.3.2. Notice that since the set A_i is initialized when L_i is created and not modified until L_{i-1} is collapsed, it is sufficient to verify the inequality after we construct the new layer $L_{\ell+1}$ in the build phase. The proof is now by contradiction. Suppose $|A_{\ell+1}| < |P_{\leq\ell}|/5$ after the build phase. Let $\mathcal{N} \subseteq \{N_1, \ldots, N_m\}$ be the clusters whose agents appear in the layered hypergraph but do not have any agents in some edge in $A_{\leq\ell+1}$. We have that, $|\mathcal{N}| = |P_{\leq\ell}| - |A_{\leq\ell+1}|$, since for any j, our algorithm introduces at most one edge into A_j for each agent in P_{j-1}.

Recall that $\mathscr{C}_t(i,\tau)$ denotes the set of those configurations for agent i, that only contain thin items. From Theorem 3.3.2 we know that there exists an x that is feasible for C-LP(τ) such that $\sum_{i\in N_k}\sum_{C\in\mathscr{C}_t(i,\tau)} x_{iC} = 1/2$ for each cluster N_k. Now, form the bipartite hypergraph $\mathscr{H} = (\mathcal{N} \cup \mathscr{R}_t, E)$ where we have vertices for clusters and thin items in $\mathscr{R}$, and edges (N_k, C) for every cluster N_k and thin configuration C (i.e., configuration consisting of thin items) pair such that $x_{pC} > 0$ and $p \in N_k$. To each edge (N_k, C) in $\mathscr{H}$ assign the weight $\left(\sum_{i\in N_k} x_{iC}\right)\sum_{j\in C} v_j$. The total weight of edges in $\mathscr{H}$ is at least $|\mathcal{N}|\tau/2$. Let Z denote the thin items appearing in the layered hypergraph and let $v(Z) = \sum_{j\in Z} v_j$ denote their value. Now, remove all items appearing in the layered hypergraph from this hypergraph to form $\mathscr{H}'$ that has edges $(N_k, C \setminus Z)$ for each edge (N_k, C) in $\mathscr{H}$. The weight of $(N_k, C \setminus Z)$ is similarly defined to be $\left(\sum_{i\in N_k} x_{iC}\right)\sum_{j\in C\setminus Z} v_j$.

Let us upper bound the total value of the thin items appearing in the layered hypergraph, Z. Consider some layer L_j. The total value of items in thin α-edges in A_j is at most $(\tau/\alpha+\tau/\beta)|A_j|$ by the minimality of thin α-edges. Next, because $B_j \subseteq M\, B_j$ contains only β-edges (see Remark 3.3.2). Therefore, the value of items in B_j not already present in some edge in A_j is at most $(\tau/\beta)|B_j|$ by minimality of the thin β-edges in B_j (see Remark 3.3.1). Therefore, $v(Z)$ is at most

$$v(Z) \leq \sum_{j=1}^{\ell}\left((\frac{\tau}{\alpha}+\frac{\tau}{\beta})|A_j| + (\frac{\tau}{\beta})|B_j|\right) + |A_{\ell+1}|\left(\frac{\tau}{\alpha}+\frac{\tau}{\beta}\right) < |A_{\leq\ell+1}|\left(\frac{\tau}{\alpha}+\frac{\tau}{\beta}\right) + |P_{\leq\ell}|\frac{\tau}{\beta}.$$

As the sum of the edge weights in $\mathscr{H}$ is at least $(|\mathcal{N}|/2)(\tau)$, the sum of edge weights in $\mathscr{H}'$ is at least $|\mathcal{N}|\tau/2 - v(Z)$. And Fact 3.3.1 implies that the sum of edge weights in $\mathscr{H}'$ must be strictly smaller than $(\mathcal{N}/2)(\tau/\alpha)$ (the contrary would imply the existence of at least one cluster whose adjacent edges in $\mathscr{H}'$ have weight at least $\frac{\tau}{2\alpha}$, which implies that there exists an adjacent edge

that contains items of total value at least τ/α). Thus,

$$\frac{(|P_{\leq\ell}| - |A_{\leq\ell+1}|)}{2}\tau - |A_{\leq\ell+1}|\left(\frac{\tau}{\alpha}+\frac{\tau}{\beta}\right) - |P_{\leq\ell}|\frac{\tau}{\beta} < \frac{(|P_{\leq\ell}| - |A_{\leq\ell+1}|)}{2}\frac{\tau}{\alpha}. \tag{*}$$

Note that $|A_{\leq\ell+1}|$ appears with a larger negative coefficient (in absolute terms) on the left-hand side than on the right-hand side. Therefore if (*) holds, then it also holds for an upper bound of $|A_{\leq\ell+1}|$. We will compute such a bound and reach a contradiction.

We start by computing an upper bound on $|A_{j+1}|$, the number of addable edges in layer L_{j+1} for $j = 0,\ldots,\ell-1$. Since layer L_j is not collapsible, it means that except for at most $\mu|P_j|$ edges in A_{j+1}, the remainder have at least $\tau/\alpha - \tau/\beta$ value of items blocked by the edges in B_{j+1}. Using this,

$$\left(\frac{\tau}{\alpha}-\frac{\tau}{\beta}\right)\left(|A_{j+1}| - \mu|P_j|\right) \leq |P_{j+1}|\frac{2\tau}{\beta} \overset{\text{summing over } j}{\Longrightarrow} \left(\frac{\tau}{\alpha}-\frac{\tau}{\beta}\right)\left(|A_{\leq\ell}| - \mu|P_{\leq\ell-1}|\right) \leq |P_{\leq\ell}|\frac{2\tau}{\beta}.$$

Rearranging terms we have,

$$|A_{\leq\ell}| \leq |P_{\leq\ell}|\frac{2\alpha}{\beta-\alpha} + \mu|P_{\leq\ell-1}| \leq |P_{\leq\ell}|\left(\frac{2\alpha}{\beta-\alpha}+\mu\right).$$

Substituting this upper bound in (*) along with our assumption $|A_{\ell+1}| < |P_{\leq\ell}|/5$ we get (after some algebraic manipulations)

$$|P_{\leq\ell}|\left(1-\frac{1}{\alpha}-\frac{2}{\beta}\right) - |P_{\leq\ell}|\left(\frac{2\alpha}{\beta-\alpha}+\mu+1/5\right)\left(1+\frac{1}{\alpha}+\frac{2}{\beta}\right) < 0.$$

This is a contradiction because if we substitute in the values of α, β, and μ the left-hand side is positive. Recall that $\alpha = 5/2, \beta = 36$, and $\mu = 1/500$. □

Now, let us comment on the implications of the above lemma. As thin items are of value less than $\tau/36$, and each edge in $A_{\leq\ell}$ is a thin α-edge of value at least $2\tau/5$, this implies that if layer L_{i-1} is not collapsible, then the number of blocking edges in L_i must be quite large. This means that the number of blocking edges will grow quickly when the layers of the layered hypergraph are not collapsible: This fact will become crucial when we design our signature vector, as it will enable us to use a logarithmic function of the sizes of the various layers as signatures.

We have the following lemma that establishes an exponential rate of growth on the number of agents per layer:

Lemma 3.3.3. *Assuming that C-LP(τ) is feasible, at the beginning of the iterative step $|P_{i+1}| > 13|P_{\leq i}|/10$ for $i = 0,\ldots,\ell-1$.*

Proof. Fix an i such that $0 \le i < \ell$. By the definition of the algorithm, L_i is not collapsible at the beginning of the iterative step. This means that there are at least $|A_{i+1}| - \mu|P_i|$ many edges in A_{i+1} that are not immediately addable. As each addable edge of A_{i+1} (except at most $\mu|P_i|$ many) has items of value at least $\tau/\alpha - \tau/\beta$ that are blocked, we can lower bound the total value of blocked items appearing in A_{i+1} by

$$\left(\frac{\tau}{\alpha} - \frac{\tau}{\beta}\right)\left(|A_{i+1}| - \mu|P_i|\right).$$

Further, since each edge in B_{i+1} is of value at most $2\tau/\beta$ by minimality, the total value of such items is upper bounded by $|P_{i+1}| \cdot 2\tau/\beta$. In total,

$$\left(\frac{\tau}{\alpha} - \frac{\tau}{\beta}\right)\left(|A_{i+1}| - \mu|P_i|\right) \le |P_{i+1}|\frac{2\tau}{\beta} \implies |P_{i+1}| \ge \frac{(\beta-\alpha)(1/5-\mu)}{2\alpha}|P_{\le i}| > 13|P_{\le i}|/10,$$

where we have used Lemma 3.3.2 to bound $|A_{i+1}|$ by $|P_{\le i}|/5$ from below. □

Since the number of blocking edges grows exponentially from layer to layer, an immediate consequence of Lemma 3.3.3 is that the total number of layers in the layered hypergraph at any step in the algorithm is at most $O(\log|\mathscr{P}|)$. This means that we have to encounter a collapse operation after at most logarithmically many iterative steps. Since our collapse operation updates a constant fraction ($\mu = 1/500$) of the agents in P_i when layer L_i is collapsed, intuitively we make large progress whenever we update M during a collapse step. We prove this by maintaining a signature vector $s := (s_0, \ldots, s_\ell, \infty)$ during the execution of the algorithm, where

$$s_i := \lfloor \log_{1/(1-\mu)} |P_i| \rfloor.$$

Lemma 3.3.4. *The signature vector always reduces in lexicographic value across each iterative step, and the coordinates of the signature vector are always non-decreasing, i.e., $s_0 \le s_1 \cdots \le s_\ell$.*

Proof. Let s and s' be the signature vectors at the beginning and at the end of some iterative step. We now consider two cases depending on whether a collapse operation occurs in this iterative step.

Case 1. **No layer was collapsed.** Clearly, $s' = (s_0, \ldots, s_\ell, s'_{\ell+1}, \infty)$ has smaller lexicographic value compared to s.

Case 2. **At least one layer was collapsed.** Let $0 \le t \le \ell$ be the index of the last layer that was collapsed during this iterative step. As a result of the collapse operation suppose the layer P_t changed to P'_t. Then we know that $|P'_t| < (1-\mu)|P_t|$. Since none of the layers with indices less than t were affected during this procedure, $s' = (s_0, \ldots, s_{t-1}, s'_t, \infty)$ where $s'_t = \lfloor \log_{1/(1-\mu)} |P'_t| \rfloor \le \lfloor \log_{1/(1-\mu)} |P_t| \rfloor - 1 = s_t - 1$. This shows that the lexicographic value of the signature vector decreases.

In both cases, the fact that the coordinates of s' are non-decreasing follows from Lemma 3.3.3 and the definition of the coordinates of the signature vector. □

Choosing the "∞" coordinate of the signature vector to be some value larger than $\log_{1/(1-\mu)} |\mathscr{P}|$ (so that Lemma 3.3.4 still holds), we see that each coordinate of the signature vector is at most U and the number of coordinates is also at most U where $U = O(\log|\mathscr{P}|)$. Thus, the sum of the coordinates of the signature vector is always upper bounded by U^2. We now prove that the number of such signature vectors is polynomial in $|\mathscr{P}|$.

A partition of an integer N is a way of writing N as the sum of positive integers (ignoring the order of the summands). The number of partitions of an integer N can be upper bounded by $e^{O(\sqrt{N})}$ by a result of Hardy and Ramanujan [20][2]. Using that the coordinates of our signature vectors are non-decreasing, each signature vector corresponds to a partition of an integer value at most U^2, and vice versa: given a partition of an integer of size ℓ, the largest number of the partition will correspond to the ℓ-th coordinate, the second largest to the $\ell-1$-th coordinate, and so on. Therefore, we can upper bound the total number of signature vectors by $\sum_{i\leq U^2} e^{O(\sqrt{i})} = |\mathscr{P}|^{O(1)}$. Since each iteration of the algorithm takes only polynomial time along with Lemma 3.3.4 this proves Lemma 3.3.1, and therefore Theorem 3.3.2.

3.4 Designing a More Efficient Algorithm

Finally, throughout this section we will design and analyze a more general polynomial-time approximation algorithm for the restricted max-min fair allocation problem. This algorithm will not make use of the clustering step we used in Section 3.3, hence it will achieve a superior approximation guarantee.

New Ideas for a Combinatorial Algorithm To begin with, let us illustrate some of the new ideas that are involved in the design of this algorithm. The first question we need to ask ourselves is: What are the shortcomings in generalizing our clustering algorithm and in enabling it to handle both fat and thin edges? In order to answer this question, we first have to consider *what options we have* with respect to doing so.

The first approach we might consider, is to directly port the clustered instances algorithm we designed to the general instances case, by simply considering the problem of assigning addable edges that can be either thin or fat to *agents*, instead of assigning thin addable edges to *clusters*. Taking this approach, however, will quickly lead to failure, once we begin analyzing the running time of our algorithm. The main problem is that now we can no longer guarantee that the number of layers is logarithmic (or even sub-polynomial). As now the layers are allowed to include fat edges, there is no way to guarantee that there will be no long paths of

[2]The asymptotic formula for the number of partitions of N is $\frac{1}{4N\sqrt{3}} \exp\left(\pi\sqrt{\frac{2N}{3}}\right)$ as $N \to \infty$ [20].

fat edges that will force the number of layers to explode, thus forcing our signature vector argument to fail (remember that our argument crucially depended on the number of layers being logarithmic in the number of clusters, which means that now it should be logarithmic in the number of agents). Consider, for example, an instance of the restricted max-min fair allocation problem, that contains a set $\mathscr{P} = T \cup F$ of m agents, where T consists of agents that can receive thin items, and F consists of agents that can receive fat items, such that $|T| = \Theta(m^{1/3})$, $|F| = \Theta(m^{2/3})$, and between any two agents $t_1, t_2 \in T$ there is a path which alternates between fat items and agents in F, of length $\Theta(m^{1/3})$. Clearly, a naive generalization of the clustered instances algorithm will have $\Omega(m^{1/3})$ layers in this case.

Therefore, if we design a polynomial-time algorithm that follows the framework we have laid out so far, this algorithm should in some sense be *oblivious to the existence of paths of fat items*, by handling them in a way that enables our attention to be focused on thin edges. For example, this was one of the achievements of Polacek and Svensson [32]: In their alternating-tree algorithm, the authors avoided having to consider fat blocking edges in their signature vector argument, by having fat edges to serve only as a way to introduce thin edges into the alternating tree, i.e., if an agent wishes to receive an addable edge, their algorithm introduces either a thin addable edge for that agent, or a path of fat edges to some other agent that then receives a thin addable edge.

Now, though our approach will move in this general direction, we have to be careful when doing so. Specifically, there will be some concerns, when we have to collapse a layer that receives enough immediately addable edges. For each such immediately addable edge that we want to introduce into our matching, we might need to use a path of fat edges in order to connect the immediately addable edge to the edge it will replace in our matching. However, there might be multiple such paths, and choosing the incorrect one might disable the inclusion of other immediately addable edges into our matching. In order to remedy this situation, we introduce the concept of *Disjoint Path Networks*: These networks will enable us to keep track of how many edges we can introduce into our matching, using paths of fat edges. Therefore, we can maximize the number of immediately addable edges we introduce into our layered hypergraph.

After the above discussion, it is time to introduce the main contribution of this section. Let $0 < \epsilon \leq 1$ be a fixed parameter, and let τ be our guess on the optimal value; we will prove the following:

Lemma 3.4.1. *There exists a polynomial-time algorithm that, given a partial matching M of agents to sets of items of value at least $\frac{\tau}{2(3+\sqrt{10})+\epsilon} > \frac{\tau}{13+\epsilon}$, extends M to include one more agent, if C-LP(τ) is feasible; the running time of this algorithm is polynomial in n and $m^{\frac{1}{\epsilon^2}\log\frac{1}{\epsilon}}$.*

As we have seen before, proving Lemma 3.4.1 is sufficient to prove Theorem 3.1.1. As in Section 3.3, we start by giving an informal overview of the algorithm behind Lemma 3.4.1. Then, we will define some necessary concepts for the algorithm's description and proceed to describe and analyze it.

3.4.1 Algorithm Overview

To begin with, the general framework of our combinatorial algorithm is similar to that of the simpler algorithm we described in Section 3.3: We guess an optimal value τ for the configuration-LP, and we then try to find an allocation of items which approximately satisfies every agent, i.e., assigns to each agent a set of items of total value at least $\tau/13$ for that agent. To do so, we will again design a local search procedure whose purpose will be to extend a given partial allocation of items, so as to satisfy one more agent.

An example execution of our combinatorial algorithm appears in Figure 3.3: There, given a partial allocation of items to agents, we want to extend this allocation to satisfy agent p. Naturally, if there is a set of items, which do not appear in the given partial allocation and whose total value for p is at least $\tau/13$, we will assign these items to agent p. Otherwise, we find an edge e_p whose total value for p is at least $\tau/2$ (the bottom gray edge in Figure 3.3(a)) and consider all the edges in our given partial allocation that share items with that set (the white edges intersecting e_p in Figure 3.3(a)); these edges constitute the first layer that is shown in Figure 3.3(a).

At this point, we should note that, as with to the simpler algorithm we described in Section 3.3, we will again be using a *greedy strategy* with respect to the edges we would include in our partial matching. Specifically, even though we want to only assign items of total value at least $\tau/13$ to each agent, the gray edges we would include in our matching are significantly more valuable (i.e., of total value at least $\tau/2$). Again, this implies that every gray edge will intersect with multiple white/blocking edges, which will help us prove that the algorithm's running time is polynomial in the size of the input.

Next, as with the simple algorithm we described in Section 3.3, we want to free up the items that appear in edge e_p. We do this by finding disjoint sets of items that satisfy the agents that appear in the white edges of the first layer. Here, however, we encounter the first major difference, compared to our previous algorithm: Some of the agents that appear in the white edges of the first layer can be satisfied by using *fat* items, i.e., items whose value for their corresponding agents is at least $\tau/13$. As every fat edge we would include in our partial allocation can only be blocked by exactly one edge that already belongs to our allocation, *alternating paths* of fat edges are created. Such a path, which ends in a gray thin edge, is displayed in Figure 3.3(b): If we are to include the gray edge that contains q_2 into our partial allocation, then we have to replace the white fat edges with the gray ones.

However, considering such alternating paths of fat edges brings up one issue: As, as is shown in Figure 3.3(a), the alternating paths that originate at agents p_1 and p_2 end at two distinct gray thin edges. Consequently, if we were to include both of these edges into our matching, then we would have to guarantee that we will not use the same fat item to satisfy two different agents. In order to do this, we will include the gray edges that contain agents q_1 and q_2 into our partial allocation, only if the alternating paths that end in these agents are *vertex-disjoint*, as is the case in Figure 3.3(c).

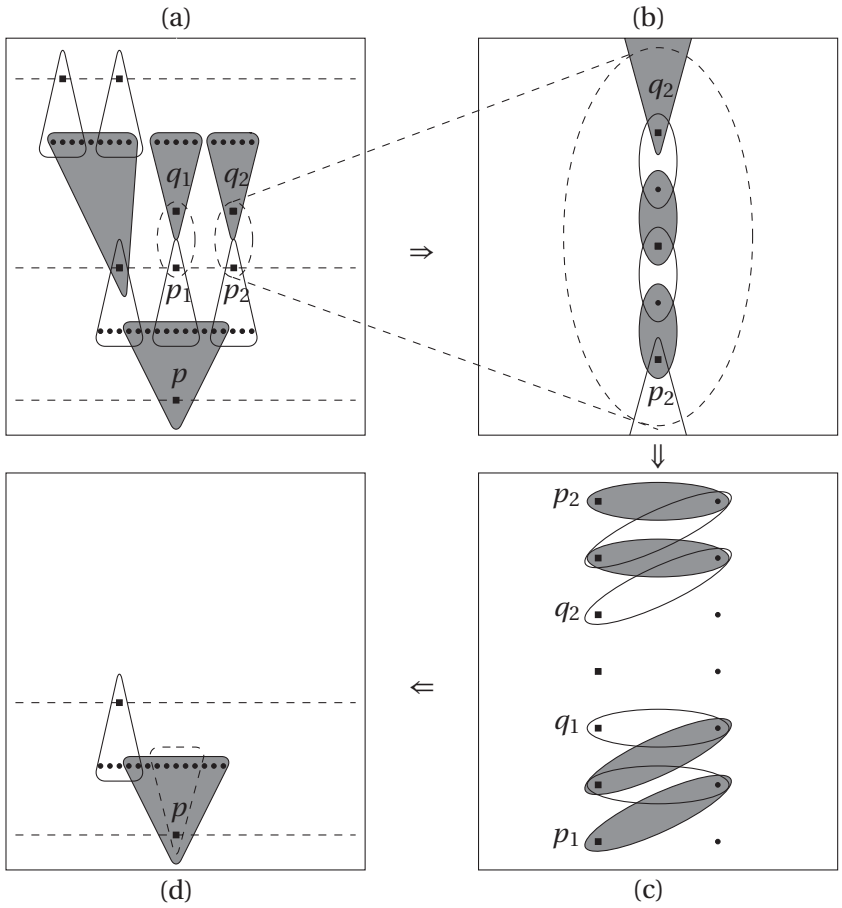

Figure 3.3: An illustration of our combinatorial algorithm. In this figure, boxes correspond to players and circles correspond to resources.

Next, as we have solved the problem of deciding *if* we can update our partial matching by replacing white edges with gray ones, the question that arises is *when* should we do this. As with our simpler algorithm, we will employ the strategy of *lazy updates*. In other words, we will replace the white edges of some layer with gray ones (we *collapse* a specific layer), only if this would mean that a significant amount of the white edges are replaced. Replacing a significant amount of white (i.e., blocking) edges then implies that we make significant progress towards matching agent p.

Finally, after we update our partial allocation, by inserting the gray edges containing agents q_1 and q_2, inserting the gray fat edges that belong to the corresponding alternating paths, and removing the white fat edges that belong to the corresponding alternating paths, we free up a significant amount of items of edge e_p. Hence, we choose a subset of the items contained in e_p, whose total value is at least $\tau/13$, and we include it in our partial allocation. At this point, we have extended our partial allocation to include one more agent, namely, agent p.

In the following section, we proceed to formally describe the algorithm behind Lemma 3.4.1,

after first defining some necessary concepts.

3.4.2 Combinatorial Algorithm

As in Section 3.3, we will design a local search algorithm whose purpose is to extend a given partial matching to include one more agent. Not surprisingly, the way our algorithm will achieve this is by constructing a layered hypergraph of addable and blocking edges. Furthermore, in order to handle fat items, our algorithm will employ disjoint path networks in order to keep track of which updates to our partial matching are possible. Before formally describing our algorithm, we first need to formalize all of the above concepts.

Parameters Throughout the description of our algorithm, we will use the following parameters:

- $\rho = 1/\beta = \frac{1}{2(3+\sqrt{10})+\epsilon}$ is the approximation guarantee of our algorithm.
- $\alpha = 2$ is the parameter that determines the how greedy our agents are.
- $\mu = \epsilon/100$ is the parameter that determines the laziness of our updates.

Thin and Fat Items In this context, we will need to redefine what a thin and a fat item (or edge) is:

Definition 3.4.1. *A* thin item *is an item whose value is at most τ/β; furthermore, we denote the thin items by $\mathcal{R}_t$, and the fat items $\mathcal{R} \setminus \mathcal{R}_t$ by $\mathcal{R}_f$.*

Definition 3.4.2. *A* thin edge *is a pair (p, R), where p is an agent and R is an inclusion-wise minimal set of thin items p is interested in, of total value at least τ/β.*

Definition 3.4.3. *A* fat edge *is a pair $(p, \{f\})$, where p is an agent and f is a fat items p is interested in.*

Definition 3.4.4. *For any $\delta \geq 1$, we call an edge (p, R) a* δ-thin edge*, or a* δ-edge*, if R is an inclusion-wise minimal set of thin items that p is interested in, of total value at least $\delta\tau$.*

Similar to Remark 3.3.1, we have the following:

Remark 3.4.1. *A δ-edge contains items of total value at most $\frac{\tau}{\delta} + \frac{\tau}{\beta}$, due to its minimality. Furthermore, given a thin β-edge (p, R), any strict subset of R contains items of total value at most τ/β.*

Similar to Section 3.3, we can now define what a partial matching is:

Definition 3.4.5. *A* partial matching *M is a set of edges of total value at least τ/β, such that no two edges in M contain intersecting items, and no two edges in M contain the same agent.*

We will say that M matches agent p if M contains an edge that includes p.

Disjoint Path Networks As we discussed in the overview of our combinatorial algorithm, we need a way to ensure that the alternating paths we use to update our partial matching are disjoint. We say that two paths are disjoint if they are vertex-disjoint. To do so, we employ a structure called *disjoint path networks*.

Given a partial matching M, let $H_M = (\mathscr{P} \cup \mathscr{R}_f, E_M)$ be the directed graph defined as follows: There is a vertex for each agent in $\mathscr{P}$ and each fat item in $\mathscr{R}_f$; and, there is an arc from an agent in $p \in \mathscr{P}$ to a fat item $f \in \mathscr{R}_f$ if p is interested in f unless the edge $(p, \{f\})$ appears in M, in which case there is an arc $(\{f\}, p)$. Note that the graph H_M depends only on the assignment of fat items to agents in M.

Now, let $S, T \subseteq \mathscr{P}$ be a set, respectively, of sources and sinks that are not necessarily disjoint. Let $\mathrm{F}_M(S,T)$ denote the flow network we obtain if we place unit capacities on the vertices of H_M; and use S and T as sources and sinks, respectively. Furthermore, let $\mathrm{DP}_M(S,T)$ denote the value of an optimal solution, i.e., the maximum number of disjoint paths from the sources S to the sinks T in the graph H_M.

In our algorithm, S and T will contain only vertices in H_M that correspond to agents in $\mathscr{P}$. However, to specify a sink we sometimes abuse notation and specify an edge as the corresponding sink vertex can be deduced from it. For example, if we write $\mathrm{DP}_M(X,Y)$, for some set of agents X and some set of edges Y, then we mean the maximum number of disjoint paths that start at an agent in X and end in an agent that appears in some edge in Y.

State of the Algorithm As in Section 3.3, we will formally define what state our algorithm is in:

Definition 3.4.6. *A state of the algorithm is a tuple $(M, p_0, \ell, \{(A_0, d_0), \ldots, (A_{\ell+1}, d_{\ell+1})\}, I)$, where:*

- *M is a partial matching.*
- *p_0 is the agent we want to match.*
- *ℓ is an index that keeps track of the depth of our search.*
- *For all $i \leq \ell + 1$, (A_i, d_i) consists of a set of α-edges A_i and a positive integer d_i.*
- *I is a set of α-edges.*

As in the algorithm in Section 3.3, the edges in some A_i correspond to edges we would like to include into our matching: Naturally, their inclusion might be blocked by edges that already exist in our partial matching. Thus, for $i > 0$, let B_i be the set of edges of M that share items with at least one edge in A_i; we conventionally set $B_0 = \{(p_0, \emptyset)\}$. Furthermore, I will contain the edges we would like to include into our matching, that contain a significant amount of items that do not appear in any edge in M (more specifically, I contains α-edges that contain items that do not appear in M of total value at least τ/β). Finally, d_i is a number that is not required to formally state our algorithm, but whose use will be convenient in our analysis; it corresponds to the number of edges in $A_{\leq i} \cup I$ that we could insert into our matching by making use of disjoint paths.

We remark that, as in the algorithm in Section 3.3, even though we define (A_i, d_i) for all $i \leq \ell + 1$, $A_{\ell+1}$ will be non-empty (and $d_{\ell+1}$ will be non-zero) only during the first phase of each iteration, and not used in the second phase of the iteration.

As in Remark 3.3.2, we maintain the following property of M:

Remark 3.4.2. *M is a partial matching that only contains β-edges and fat edges.*

Using the definition of the state of the algorithm, we can now define what is a layer in our combinatorial algorithm:

Definition 3.4.7. *Given a state of the algorithm, we let $L_i = (A_i, B_i, d_i)$ denote the i-th* layer *of our algorithm. Furthermore, let $\mathscr{L} = \{L_0, \ldots, L_{\ell+1}\}$.*

Naturally, we will call $\mathscr{L}$ the *layered hypergraph* maintained by our algorithm.

Canonical Decompositions We proceed to define the next concept necessary for describing our combinatorial algorithm. Recall that we denote $\cup_{i \leq t} S_i$ by $S_{\leq t}$, for some sequence of sets $S_0, \ldots S_t$, and that P_i denotes the agents that appear in B_i. Moreover, for set S of edges, we use $\mathscr{P}(S)$ to denote the set of agents that appear in an edge in S, and we use $\mathscr{R}(S)$ to denote the set of items that appear in a set of edges S.

Definition 3.4.8 (Canonical Decomposition of I)**.** *Consider some state of the algorithm $(M, p_0, \ell, \{(A_0, d_0), \ldots, (A_{\ell+1}, d_{\ell+1})\}, I)$. We call a collection of disjoint subsets $\{I_0, I_1, \ldots, I_\ell\}$ of I a* canonical decomposition *if it satisfies the following conditions:*

1. *For $i = 0, 1, \ldots, \ell$, $|I_{\leq i}| = \mathrm{DP}_M(P_{\leq i}, I_{\leq i}) = \mathrm{DP}_M(P_{\leq i}, I)$.*
2. *There exists an optimal solution W to $\mathrm{F}_M(P_{\leq \ell}, I)$ such that, for $i = 0, 1 \ldots, \ell$, $|I_i|$ paths in W go from agents $Q_i \subseteq P_i$ to the sinks in I_i. We denote these paths by W_i. We also refer to W as the* canonical solution *corresponding to the decomposition.*

As we will see later on, canonical decompositions and solutions can be computed in polynomial time.

The Algorithm for General Instances As in Section 3.3, we proceed to define what an addable, immediately addable, and blocking edge is:

Definition 3.4.9. *Given a state of the algorithm, we call an α-edge $e = (p, R)$* addable*, if R does not intersect any edge in $A_{\leq \ell+1} \cup B_{\leq \ell} \cup I$ and $\mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell+1} \cup I \cup \{e\}) > \mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell+1} \cup I)$.*

Definition 3.4.10. *Given a state of the algorithm, we call an addable edge $e = (p, R)$* immediately addable*, if there exists $R' \subseteq R$ such that the total value of items in R' is at least τ/β, and none of the items in R' appear in any edge in M.*

Definition 3.4.11. *Given a state of the algorithm, we call an edge $e \in M$ that belongs to some B_i a* blocking *edge; similarly, given an edge e' that belongs to some A_i and an edge $e \in M$ that shares items with e', we will say that e* blocks *e'.*

Given the above definitions, we can define what a collapsible layer is:

Definition 3.4.12. *Given a state of the algorithm $(M, p_0, \ell, \{(A_0, d_0), \ldots, (A_{\ell+1}, d_{\ell+1})\}, I)$, and a canonical decomposition $\{I_0, I_1, \ldots, I_\ell\}$, we call a layer L_i* collapsible*, if $|I_i| \geq \mu |P_i|$.*

Finally, we are ready to describe the algorithm behind Lemma 3.4.1: This algorithm takes a partial matching M and augments it to one of larger size. We start with a partial matching M that assigns the maximum possible number of fat edges. Note that this can be done in polynomial time by simply solving the maximum matching problem in the bipartite graph where one partition corresponds to $\mathscr{P}$ and the other to $\mathscr{R}_f$, and edges signify an agent is interested in a fat item. The input to our algorithm is then such a partial matching M.

Now, when we want to extend M, we do the following:

Initialization: Select an agent p_0 not matched by M. The goal is to extend M so as to also match p_0 in addition to the agents already matched by M. Set $I \leftarrow \emptyset$, $A_0 \leftarrow \emptyset$ and $d_0 \leftarrow 0$. The first layer L_0 is now defined by the tuple (A_0, B_0, d_0). Set $\ell \leftarrow 0$.

Iterative step: The iterative step consists of two phases, the first of which is always executed:

Build Phase: Set $A_{\ell+1} \leftarrow \emptyset$. While there exists an addable edge e, we add it to I if e is immediately addable, otherwise to $A_{\ell+1}$. When this is no longer possible, set $d_{\ell+1} \leftarrow \mathrm{DP}_M(P_{\leq l}, A_{\leq l+1} \cup I)$. Now set the new layer $L_{\ell+1} \leftarrow (A_{\ell+1}, B_{\ell+1}, d_{\ell+1})$, increment ℓ and proceed to the next phase of the iteration.

Collapse Phase: Compute the canonical decomposition $I_0 \cup \cdots \cup I_\ell$ of I. While there exists a collapsible layer, let L_t be the earliest collapsible layer and do the following:

i. Compute the optimal solution W corresponding to the canonical decomposition of I and compute an optimal solution X to $\mathrm{F}_M(P_{\leq t-1}, A_{\leq t} \cup I_{\leq t-1})$ whose paths are disjoint from W_t, where W_t are the $|I_t|$ paths that go from the agents in $Q_t \subseteq P_t$ to sinks in I_t. For each path Π in W_t that ends at an agent p_e with an edge $(p_e, R) \in I_t$:

(Alternating along path Π*)* A. Set $M \leftarrow M \setminus \{(p,\{f\}) \mid (f,p) \in \Pi\} \cup \{(p,\{f\}) \mid (p,f) \in \Pi\}$.

B. Remove from M the edge containing the source of the path Π.

C. Add to M some β-edge (p_e, R'), where $R' \subseteq R$ is a set of thin items that does not appear in any edge in M.

ii. Set I to be $I_0 \cup \cdots \cup I_{t-1}$. For each edge $a = (p,R) \in A_t$ that contains items of total value at least τ/β that do not appear in any edge in M, set $A_t \leftarrow A_t \setminus \{a\}$; then, if X has a path that ends at a, set $I \leftarrow I \cup \{a\}$[3].

iii. Discard all the layers with index greater than t and set ℓ to be t. Repeat the collapse phase until there are no collapsible layers.

Repeat the iterative step until p_0 is matched by M.

Similar to the algorithm we introduced in Section 3.3, our combinatorial algorithm preserves the following invariants:

1. For $i = 0,\ldots,\ell$, A_i is a set of thin α-edges and each α-edge $(p,R) \in A_i$ has $R \cap \mathscr{R}(A_{\leq i} \cup B_{\leq i-1} \cup I \setminus \{(p,R)\}) = \emptyset$ (its items are not shared with edges from earlier iterations, edges in A_i, or edges in I).

2. For any edge $(p,R) \in I$, it holds that $R \cap \mathscr{R}(A_{\leq \ell} \cup I \setminus \{(p,R)\}) = \emptyset$ and R contains items of total value at least τ/β that do not appear in M.

The similarities between these invariants and those of the simpler algorithm follow from the same basic ideas. However, as the combinatorial algorithm we present in this section is more involved, its analysis requires more invariants that we present in the subsequent sections.

Before proceeding with the analysis of our combinatorial algorithm, we explain its steps in more detail and why the algorithm satisfies the above invariants. The algorithm begins with a partial matching M and an agent p_0 that we want to include in our partial matching. Furthermore, we make sure that M contains a maximum matching between fat items and agents. Every iteration of our algorithm involves two main phases: the build phase, and the collapse phase.

During the build phase of layer $\ell+1$, the algorithm finds, for the agents in P_ℓ, thin addable α-edges that we then insert into either I (if the edge is immediately addable) or to $A_{\ell+1}$. By the design of our combinatorial algorithm, any edge that is inserted into $A_{\ell+1}$ will be disjoint from edges in $A_{\leq \ell+1} \cup B_{\leq \ell} \cup I$; the same holds for any edge (p,R) that is inserted into I, while in addition we have items of total value at least τ/β that are disjoint from the items appearing in edges in M. Therefore, the two invariants are preserved during the build phase.

[3] Notice that, after being removed from A_t and before being inserted into I, a actually satisfies the conditions of an *immediately addable* edge.

Furthermore, edges inserted into $A_{\ell+1}$ or I need either to contain an agent from $P_{\leq \ell}$, or to be the final edge in an alternating path that includes fat edges originating at an agent in $P_{\leq \ell}$. Even though we will not store such alternating paths explicitly, according to the definition of addable edges, it is required that after we insert any such thin α-edge into $A_{\ell+1}$ or I, the value of the flow network $\mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell+1} \cup I)$ be increased. This will ensure that there are enough disjoint paths of fat edges to permit the inclusion of all such thin edges into our partial matching M.

After the algorithm has finished the build phase, it proceeds to the collapse phase. We will describe this phase in more detail and show that it maintains a valid matching and that it does not introduce any violations of the two invariants.

The first step of the collapse phase is to compute a canonical decomposition of I and a corresponding canonical solution W. Suppose that we have $|I_t| \geq \mu |P_t|$ and that the algorithm collapses layer t. The edges in I_t are the edges that we, using the paths of W_t, will insert into our partial matching. Specifically, for each path Π of W_t, the algorithm proceeds as follows. By definition of the sources and the sinks, Π is a path that starts with an agent $p_s \in P_t$ and ends with an agent p_e such that $(p_e, R) \in I_t$. Between p_s and p_e, the path alternates between fat edges that belong to M and fat edges we want to insert into M, i.e., $\Pi = (p_s = p_1, f_1, p_2, f_2, \ldots, p_k, f_k, p_{k+1} = p_e)$ where p_s is interested in f_1, p_{k+1} is currently assigned f_k, and p_i is currently assigned f_{i-1} and interested in f_i for $i = 2, \ldots, k$. To update the matching, we find a β-edge (p_e, R') with $R' \subseteq R$ that is disjoint from the items of matching M (guaranteed to exist by the second invariant) and we let (p_s, R_s) denote the edge in $B_t \subseteq M$ incident to agent p_s. Step (i) of the collapse phase then updates the matching by inserting (p_e, R') and $(p_s, f_1), (p_2, f_2), \ldots, (p_k, f_k)$ to the matching while removing (p_s, R_s) and $(p_2, f_1), (p_3, f_2), \ldots, (p_t, f_k)$. This process is called *alternating along path* Π.

As a result of the collapse phase, some of the items of edges in A_t are freed up, and we move these edges of A_t that have τ/β free items to I (Step (ii)). Finally, we discard all layers above the one we collapsed. Let us now see why our first invariant is upheld after the collapse phase. When we collapse layer t, we might remove edges from A_t, we discard all $A_{t'}$ for $t' > t$ and we preserve $A_{t'}$ for $t' < t$. As the first invariant was upheld before the collapse phase, for any $t' \leq t$ there were no edges in $A_{t'}$ that intersected any edge in $A_{\leq t'}$, $B_{\leq t'-1}$ or $I_0 \cup \ldots I_{t-1}$. Furthermore, as any edge that was inserted into I during Step (ii) previously belonged to A_t, no edge inserted into I will intersect any edge in $A_{\leq t} \cup B_{\leq t-1} \cup I_0 \cup \ldots I_{t-1}$. Therefore, after the collapse phase, for any $t' \leq t$ every edge in $A_{t'}$ is disjoint from edges in $A_{\leq t'} \cup B_{\leq t'-1} \cup I$, and the first invariant holds.

After the collapse phase, I contains the edges that belonged to $I_0 \cup \ldots I_{t-1}$ (call them *old* edges), plus the edges that were inserted during Step (ii) (call them *new* edges). Concerning any old edge e, as the second invariant held before the collapse phase, and as during the collapse phase for any $t' \leq t$ we introduce no new edges into $A_{t'}$, the items of e continue to be disjoint from the items of $A_{\leq t}$ and the old edges. Moreover, e still contains items that, of total value at

least τ/β, are disjoint from the items in M, as the items of the edges added to the matching during the collapse phase are disjoint from $\mathscr{R}(e)$, where we use that the second invariant held before this iteration, i.e., that the items of edges in I are disjoint. Hence, to verify the second invariant we need to verify that any new edge (p,R) has items, of total value at least τ/β, disjoint from the items in M (this follows immediately from Step (ii) of the collapse phase) and to verify that its items are disjoint from the items of all old and other new edges and edges in $A_{\leq t}$. This follows directly from the fact that any new edge belonged to A_t before the collapse phase and that the first invariant held before the collapse phase. Hence, the second invariant is satisfied after the collapse phase.

Now, let us see why the output of our combinatorial algorithm is a partial matching that matches agent p_0. Observe that we only update our partial matching during Step (i) and, as explained above, we alternate along all paths in W_t during this step. As these paths are vertex-disjoint and the edges in I have disjoint items (by the second invariant), these updates do not interfere with each other. Moreover, note that when we alternate along a path, all previously matched agents remain matched (albeit to new edges), and all fat items remain matched. This means that our algorithm maintains a matching of the agents that were matched by the input, and that this matching remains one of the many that maximize the number of assigned fat items. By iterating until an edge that contains p_0 is inserted into M, it follows that when our combinatorial algorithm terminates, the output will be a valid matching that also matches p_0, in addition to the agents that were matched by the original matching that was given as input.

In the subsequent sections, we prove that the running time of each iteration is polynomial, and that the total number of executed iterations is also polynomial.

3.4.3 Running-Time Analysis of each Iteration

We have yet to prove that if C-LP(τ) is feasible, the above algorithm extends M in polynomial time. First of all, let us prove that every single iteration can be executed in polynomial time. We begin by studying the build phase. In this phase, in each iteration of the while-loop, we consider an addable edge (p,R). In doing so, we need to check whether adding p as a sink to our flow network strictly increases its value, i.e., if the number of disjoint paths from the sources in $P_{\leq \ell}$ to the sinks in $A_{\leq \ell+1} \cup I \cup \{(p,R)\}$ increases. Both these operations can be done in polynomial time: Because (1) verifying whether such a set R exists for an agent p amounts to simply calculating the total value of the items, currently not in the other relevant edges, that p is interested in; and as (2) verifying whether the flow network increases its value reduces to a standard maximum flow problem.

Next, we study the collapse operation. Here, we have two non-trivial operations: computing a canonical decomposition at the beginning of the collapse phase, and Step (i) of the collapse phase:

Lemma 3.4.2. *Given a state $(M, p_0, \ell, \{(A_0,d_0),\ldots,(A_{\ell+1},d_{\ell+1})\}, I)$ of the algorithm, we can find a canonical decomposition of I and the corresponding canonical solution in polynomial*

time.

Proof. We construct an optimal solution W to the flow network $\mathrm{F}_M(P_{\leq \ell}, I)$ with sources $P_{\leq \ell}$ and sinks I iteratively. We compute the maximum flow $X^{(0)}$ in the network $\mathrm{F}_M(P_{\leq 0}, I)$. Let $Q_0 \subseteq P_0$ be the set of sources appearing in the flow solution $X^{(0)}$. Observe that this solution $X^{(0)}$ is also a valid flow in the network $\mathrm{F}_M(P_{\leq 1}, I)$. Therefore, by using an augmenting flow algorithm, we can augment the flow $X^{(0)}$ to a maximum flow $X^{(1)}$ in the network $\mathrm{F}_M(P_{\leq 1}, I)$. Let $Q_1 \subseteq P_1$ be the set of additional sources appearing in the flow solution $X^{(1)}$. We use here an important property of the flow augmentation process, that states that the set of sources in $X^{(1)}$ is precisely the disjoint union $Q_0 \cup Q_1$. In other words, a vertex, appearing as a source of a flow path in a solution, continues to be present as a source of a flow path after an augmentation step. Continuing this process, we end up with a flow solution $X^{(\ell)}$ in the network $\mathrm{F}_M(P_{\leq \ell}, I)$. Let W_i be the flow paths in $X^{(\ell)}$ that serve the sources $Q_i \subseteq P_i$ for each $i = 0, \ldots, \ell$. Additionally, let $I_i \subseteq I$ denote the sinks of W_i.

By construction, $|I_{\leq i}| = \mathrm{DP}_M(P_{\leq i}, I_{\leq i})$. Further, if $\mathrm{DP}_M(P_{\leq i}, I_{\leq i}) < \mathrm{DP}_M(P_{\leq i}, I)$ then this implies that $X^{(i)}$ is not a maximum flow in $\mathrm{F}_M(P_{\leq i}, I)$. Hence, it can be augmented by one, thus contradicting the definition of $X^{(i)}$.

The flow paths $W_0, W_1, \ldots, W_\ell$ collectively form the flow solution $X^{(\ell)}$ that is an optimal solution to $\mathrm{F}_M(P_{\leq \ell}, I)$. Thus, $\{I_0, \ldots, I_\ell\}$ forms a canonical decomposition (with the corresponding canonical solution $W_0, \ldots, W_\ell$). It is also clear that the process outlined above, for the realization of this decomposition, runs in polynomial time as the encountered flow networks have unit capacities. □

Next, we prove that Step (i) of the collapse phase can be executed in polynomial time:

Lemma 3.4.3. *Consider a state $(M, p_0, \ell, \{(A_0, d_0), \ldots, (A_{\ell+1}, d_{\ell+1})\}, I)$ of the algorithm and a canonical decomposition $\{I_0, I_1, \ldots, I_\ell\}$ of I together with the canonical solution W. For $i = 0, \ldots, \ell$, let W_i be the $|I_i|$ paths that go from the agents in $Q_i \subseteq P_i$ to sinks in I_i. Then, for $i = 0, 1, \ldots, \ell - 1$, we can find in polynomial time an optimal solution X to $\mathrm{F}_M(P_{\leq i}, A_{\leq i+1} \cup I_{\leq i})$ that is also an optimal solution to $\mathrm{F}_M(P_{\leq i}, A_{\leq i+1} \cup I)$ whose paths are disjoint from the paths in W_{i+1} and additionally uses all the sinks in $I_{\leq i}$.*

Proof. Consider a fixed i. We form an optimal solution X to $\mathrm{F}_M(P_{\leq i}, A_{\leq i+1} \cup I_{\leq i})$; it is also an optimal solution to $\mathrm{F}_M(P_{\leq i}, A_{\leq i+1} \cup I)$, its paths are disjoint from the paths in W_{i+1} and it uses all the sinks in $I_{\leq i}$. The initial solution will be the set of unit flow paths $W_{\leq i}$ from the canonical solution W that has cardinality $|I_{\leq i}|$. We now augment this solution by using augmenting paths to the set of sinks $A_{\leq i+1}$. As, throughout this execution, each vertex in $I_{\leq i}$ will be used as a sink by some path, X will use all these sinks. Further, the procedure to calculate X clearly runs in polynomial time. We verify the remaining properties of X. First, suppose towards contradiction that some iteration used an augmenting path P intersecting a path in W_{i+1}. However, this would imply that there exists an augmenting path that uses a sink

in I_{i+1}. We could then increase the set of disjoint paths from agents in $P_{\leq i}$ to sinks in I to be greater than $I_{\leq i}$ which contradicts the property $\mathrm{DP}_M(P_{\leq i}, I_{\leq i}) = \mathrm{DP}_M(P_{\leq i}, I)$ of the canonical decomposition. Similarly, suppose X is not an optimal solution to $\mathrm{F}_M(P_{\leq i}, A_{\leq i+1} \cup I)$. Then there exists an augmenting path to an edge in $I \setminus I_{\leq i}$ which again contradicts the property $\mathrm{DP}_M(P_{\leq i}, I_{\leq i}) = \mathrm{DP}_M(P_{\leq i}, I)$ of the canonical decomposition. □

Finally, as during a collapse operation we can collapse at most $|\mathscr{P}|$ layers, it follows that any iteration of our combinatorial algorithm terminates in polynomial time.

3.4.4 Additional Invariants of Combinatorial Algorithm

In Section 3.4.2 we listed, along with the original description of our algorithm, certain invariants our combinatorial algorithm preserves that are similar to the simpler algorithm. We argued why they hold, and how these invariants imply that the output of our algorithm is an extended partial matching. In this section, we list two new invariants that will facilitate our polynomial running-time proof.

Lemma 3.4.4. *At the beginning of each iteration:*

1. $\mathrm{DP}_M(P_{\leq \ell}, I) = |I|$.
2. $\mathrm{DP}_M(P_{\leq i-1}, A_{\leq i} \cup I) \geq d_i$ *for each* $i = 1, \ldots, \ell$.

Proof. We prove the lemma by induction on the number of times the iterative step has been executed. We observe that both invariants trivially hold before the first execution of the iterative step. We assume that they are true before the r-th execution of the iterative step. We now verify them before the $r+1$-th iterative step. We technically prove the stronger statement that they hold after the build phase and after each iteration of the collapse phase.

(1) and (2) hold after the build phase. Let $L_{\ell+1}$ denote the layer that was constructed during the build phase. We start by verifying (1). If no edge is added to I during this phase, then $|I| \geq \mathrm{DP}_M(P_{\leq \ell+1}, I) \geq \mathrm{DP}_M(P_{\leq \ell}, I) = |I|$. We suppose that $a_1, \ldots, a_k$ were the edges added to the set I in that order. When edge a_i was added to the set I, from the definition of addable edges we have that

$$\mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell} \cup I \cup \{a_1, \ldots, a_{i-1}\} \cup \{a_i\}) > \mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell} \cup I \cup \{a_1, \ldots, a_{i-1}\}),$$

which then implies that

$$\mathrm{DP}_M(P_{\leq \ell}, I \cup \{a_1, \ldots, a_{i-1}\} \cup \{a_i\}) > \mathrm{DP}_M(P_{\leq \ell}, I \cup \{a_1, \ldots, a_{i-1}\})$$

To see this implication, observe that the first inequality implies that, for any flow in the network $\mathrm{F}_M(P_{\leq \ell}, A_{\leq \ell} \cup I \cup \{a_1, \ldots, a_{i-1}\})$ (hence, for any flow in $\mathrm{F}_M(P_{\leq \ell}, I \cup \{a_1, \ldots, a_{i-1}\})$), there exists

an augmenting path towards sink a_i. Along with the induction hypothesis, these inequalities imply that

$$\mathrm{DP}_M(P_{\leq \ell+1}, I \cup \{a_1, \ldots, a_k\}) \geq \mathrm{DP}_M(P_{\leq \ell}, I \cup \{a_1, \ldots, a_k\}) = |I| + k = |I \cup \{a_1, \ldots, a_k\}|.$$

For (2), the inequality for $i = \ell + 1$ holds by the definition of $d_{\ell+1}$ during this phase. The remaining inequalities follow from the induction hypothesis since none of $M, P_{\leq \ell}$ and $A_{\leq \ell}$ were altered during this phase and no elements from I were discarded.

(1) and (2) hold after each iteration of the collapse phase. If no layer is collapsed (i.e., there is no I_t satisfying the condition of the while-loop), then there is nothing to prove. Now let t denote the index of the layer that is collapsed. Let $(M, p_0, \ell, \{(A_0, d_0), \ldots, (A_{t'}, d_{t'})\}, I)$ denote the state of the algorithm before collapsing layer t that satisfy (1) and (2) ($t' \geq t$ and $t' = \ell + 1$ if this is the first iteration of the collapse phase). Let I' denote $I_0 \cup \cdots \cup I_{t-1} \cup \{a_1, \ldots, a_k\}$ where $a_1, \ldots, a_k$ are the edges added to I in Step (ii) of the collapse phase and let M' denote the partial matching after Step (i.C) of the collapse phase. We have that (1), $\mathrm{DP}_{M'}(P_{\leq t}, I') = |I'|$, now follows from Lemma 3.4.3. Indeed, the solution X used all the sinks in $I_0 \cup \ldots I_{t-1} \cup \{a_1, \ldots, a_k\}$ which equals I'; and these paths form a solution to $\mathrm{F}_{M'}(P_{\leq t}, I')$ as they are disjoint from the paths in W_t. Notice that we do not use the induction hypothesis in this case, i.e., that $(M, \ell, \{L_0, \ldots, L_{t'}\}, I)$ satisfied (1) and (2).

For (2), we need to verify inequalities for $i = 1, \ldots, t$. When $i < t$, none of the sets A_i were altered during this iterative step. Furthermore, although M and I change during the collapse phase, according to Lemma 3.4.3 and the definition of the collapse phase, this change cannot reduce the number of disjoint paths from $P_{\leq i-1}$ to $A_{\leq i} \cup I$ and therefore (2) remains true by the induction hypothesis. Indeed, we select X in Step (i) of the collapse phase so as to make sure that the update of the matching along the alternating paths in W_t does not interfere with an optimal solution to the flow network with sources $P_{\leq i-1}$ and sinks $A_{\leq i} \cup I$. For $i = t$, the claim again follows since the number of disjoint paths from $P_{\leq t-1}$ to $A_{\leq t} \cup I$ cannot reduce because of Step (ii) of the collapse phase in the algorithm that maintains X as a feasible solution by the same arguments as for (1). □

3.4.5 Bounding the Total Number of Iterations

In this final section, we will use the above invariants to show that our augmenting algorithm performs a polynomial number of iterations, assuming C-LP(τ) is feasible. We begin with two lemmas that show that d_i cannot be too small. The first holds in general, and the second holds if C-LP(τ) is feasible.

Lemma 3.4.5. *At the beginning of each iteration, we have $d_i \geq |A_{\leq i}|$ for every $i = 0, \ldots, \ell$.*

Proof. We prove this by induction on the variable $r \geq 0$ that counts the number of times the iterative step has been executed. For $r = 0$ the statement is trivial. Suppose that it is true

for $r \geq 0$. We shall show that it holds before the $r+1$-th iterative step. If the r-th iteration collapses a layer, then no new layer was added, and since d_i's remain unchanged and $A_{\leq i}$ can only decrease, the statement is true in this case.

Now, suppose that no layer was collapsed in this iteration and let $L_{\ell+1} = (A_{\ell+1}, B_{\ell+1}, d_{\ell+1})$ be the newly constructed layer in this phase. Again, we have $d_i \geq |A_i|$ for $i = 0, \ldots, \ell$ since none of these quantities are changed by the build phase. Let us now verify that $d_{\ell+1} \geq A_{\ell+1}$. Let $A_{\ell+1} = \{a_1, \ldots, a_k\}$ denote the set of edges added to $A_{\ell+1}$ indexed by the order in which they were added. When edge a_i was added to the set $A_{\ell+1}$, according to the build phase of our combinatorial algorithm, we have that

$$\mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell} \cup I \cup \{a_1, \ldots, a_{i-1}\} \cup \{a_i\}) > \mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell} \cup I \cup \{a_1, \ldots, a_{i-1}\}).$$

Using (2) of Lemma 3.4.4 and the induction hypothesis,

$$\mathrm{DP}_M(P_{\leq \ell-1}, A_{\leq \ell} \cup I) \geq d_\ell \geq |A_{\leq \ell}|.$$

Using the previous inequalities,

$$d_{\ell+1} = \mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell+1} \cup I) \geq |A_{\leq \ell}| + k \geq |A_{\leq \ell+1}|.$$

□

Lemma 3.4.6. *Assuming C-LP(τ) is feasible, at the beginning of each iteration*

$$\mathrm{DP}_M(P_{\leq i-1}, A_{\leq i} \cup I) \geq d_i \geq \gamma |P_{\leq i-1}|, \text{ where } \gamma = \frac{1}{3}(\sqrt{10} - 2),$$

for every $i = 1, \ldots, \ell$.

Proof. We will prove that $d_i \geq \gamma |P_{\leq i-1}|$ for $i = 1, \ldots, \ell$ as Lemma 3.4.4(2) then implies the claim. Notice that d_i is defined only at the time when layer L_i is created and not altered thereafter. So it suffices to verify that, assuming $d_i \geq \gamma |P_{\leq i-1}|$ for $i = 1, \ldots, \ell$, then for the newly constructed layer $L_{\ell+1}$, $d_{\ell+1} \geq \gamma |P_{\leq \ell}|$ also.

Suppose towards contradiction that $L_{\ell+1}$ is a newly constructed layer (and that no layer was collapsed), such that

$$d_{\ell+1} = \mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell+1} \cup I) < \gamma |P_{\leq \ell}|.$$

Then, as no layer was collapsed during the collapse phase of our algorithm, we have that $|I_i| < \mu |P_i|$ for $i = 0, \ldots, \ell$, where $\{I_0, \ldots, I_\ell\}$ is the canonical decomposition of I considered by the algorithm. Together with Lemma 3.4.4(1), this implies

$$|I| = \mathrm{DP}_M(P_{\leq \ell}, I) < \mu |P_{\leq \ell}|.$$

Moreover, by Lemma 3.4.5 we have

$$|A_{\leq \ell+1}| \leq d_{\ell+1} = \mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell+1} \cup I) < \gamma |P_{\leq \ell}|.$$

Hence, we have that $|A_{\leq \ell+1} \cup I| < (\mu + \gamma)|P_{\leq \ell}|$.

We devote the rest of the proof to showing that this causes the dual of the C-LP(τ) to become unbounded, which leads to the required contradiction by weak duality. That is, we can then conclude that if C-LP(τ) is feasible then $d_{\ell+1} \geq \gamma |P_{\leq \ell}|$.

Consider the flow network $\mathrm{F}_M(P_{\leq \ell}, A_{\leq \ell+1} \cup I \cup Z)$ with $P_{\leq \ell}$ as the set of sources and $A_{\leq \ell+1} \cup I \cup Z$ as the collection of sinks where,

$$Z := \{p \in \mathscr{P} \mid \exists R \subseteq \mathscr{R} \,:\, R \cap \mathscr{R}(A_{\leq \ell+1} \cup I \cup B_{\leq \ell}) = \emptyset \text{ and } \sum_{j \in R} v_{pj} \geq \tau/\alpha\}.$$

Since, during the construction of layer $\ell+1$ we could not insert any more edges into $A_{\ell+1}$ and I, the maximum number of vertex disjoint paths from $P_{\leq \ell}$ to the sinks equals $\mathrm{DP}_M(P_{\leq \ell}, A_{\leq \ell+1} \cup I)$ which, by assumption, is less than $\gamma |P_{\leq \ell}|$. Therefore, by Menger's theorem there exists a set $K \subseteq V$ of vertices of cardinality less than $\gamma |P_{\leq \ell}|$ such that, if we remove K from H_M, the sources $P_{\leq \ell} \setminus K$ and the sinks are disconnected, i.e., no sink is reachable from any source in $P_{\leq \ell} \setminus K$. We now claim that we can always choose such a vertex cut so that it is a subset of the agents.

Claim 3.4.1. *There exists a vertex cut $K \subseteq \mathscr{P}$ separating $P_{\leq \ell} \setminus K$ from the sinks of cardinality less than $\gamma |P_{\leq \ell}|$.*

Proof. Take any minimum cardinality vertex cut K separating $P_{\leq \ell} \setminus K$ from the sinks. We already saw that $|K| < \gamma |P_{\leq \ell}|$. Observe that every fat item that is reachable from $P_{\leq \ell} \setminus K$ must have outdegree exactly one in H_M. It cannot be more than one since M is a collection of disjoint edges, and it cannot be zero since we could then increase the number of fat edges in M, which contradicts that we started with a partial matching that maximized the number of fat edges. Therefore in the vertex cut K, if there are vertices corresponding to fat items, we can replace each fat item with the unique agent to which it has an outgoing arc, in order to obtain another vertex cut also of the same cardinality that contains only vertices corresponding to agents. □

Now call the induced subgraph of $H_M - K$ on the vertices that are reachable from $P_{\leq \ell} \setminus K$ as H'. Note that by the definition of K, H' will not contain any sinks. Using H', we define the assignment of values to the dual variables in the dual of C-LP(τ) as follows:

$$y_i := \begin{cases} (1-1/\alpha) & \text{if agent } i \text{ is in } H', \\ 0 & \text{otherwise,} \end{cases}$$

$$z_j := \begin{cases} v_j/\tau & \text{if } j \text{ is a thin item that appears in } A_{\leq \ell+1} \cup I \cup B_{\leq \ell}, \\ (1-1/\alpha) & \text{if } j \text{ is a fat item in } H', \\ 0 & \text{otherwise.} \end{cases}$$

We first verify that the above assignment is feasible. Since all the dual variables are non-negative we only need to verify that $y_i \leq \sum_{j \in C} z_j$ for every $i \in \mathscr{P}$ and $C \in \mathscr{C}(i,\tau)$. Consider an agent i that is given a positive y_i value by the above assignment. Let $C \in \mathscr{C}(i,\tau)$ be a configuration for agent i of value at least τ; we will call C thin if it only contains thin items, and fat otherwise. There are two cases we need to consider.

Case 1. **C is a thin configuration.** Suppose that $\sum_{j \in C} z_j < (1-1/\alpha)$. Then, by our assignment of z_j values, this implies that there exists a set $R \subseteq C$ such that R is disjoint from the items in $A_{\leq \ell+1} \cup I \cup B_{\leq \ell}$ and $\sum_{j \in R} v_j \geq \tau/\alpha$. Together this contradicts the fact that H' has no sinks, since i is then a sink (it is in Z).

Case 2. **C is a fat configuration.** Let j be a fat item in C. Since i was reachable in H', all the sources in H' are assigned thin edges in M (which implies they have no incoming arcs), and K is a subset of the agents, it follows that j is also present in H'. Thus, by our assignment, $z_j = 1-1/\alpha$.

Having proved that our assignment of y_i and z_j values constitutes a feasible solution to the dual of C-LP(τ), we now compute the objective function value $\sum_i y_i - \sum_j z_j$ of the above assignment. To do so, we adopt the following charging scheme: for each fat item j in H', we charge its z_j value against the unique agent i such that the outgoing arc (j,i) belongs to H'. The charging scheme accounts for the z_j values of all the fat items except for the fat items that are leaves in H'. There are at most $|K_1|$ such fat items, where $K_1 \subseteq K$ is the set of agents to which the uncharged fat items have an outgoing arc. Moreover, note that K_1 only consists of agents that are matched in M by fat edges. Since $P_{\leq \ell}$ does not have any agents matched by fat edges in M, no agent in $K_2 := P_{\leq \ell} \cap K$ is present in K_1, i.e., $K_1 \cap K_2 = \emptyset$. Finally, note that no agent in $P_{\leq \ell} \setminus K = P_{\leq \ell} - K_2$ has been charged. Thus, considering all agents in $\mathscr{P}$ but only fat configurations, we have

$$\begin{aligned} \sum_{i \in \mathscr{P}} y_i - \sum_{j \in \mathscr{R}_f} z_j &\geq (1-1/\alpha)(|P_{\leq \ell}| - |K_2|) - (1-1/\alpha)|K_1| \\ &= (1-1/\alpha)\big(|P_{\leq \ell}| - (|K_1| + |K_2|)\big) \\ &> (1-1/\alpha)(1-\gamma)|P_{\leq \ell}|. \end{aligned}$$

We now compute the total contribution of thin items, i.e., $\sum_{j\in\mathscr{R}_t} z_j$. The total value of thin items from the edges $A_{\leq \ell+1}$ and the edges I is at most $(1/\alpha + 1/\beta)|A_{\leq \ell+1} \cup I|$, due to the minimality of thin α-edges. Whereas, since $B_{\leq \ell} \subseteq M$, $B_{\leq \ell}$ only contains β-edges (see Remark 3.4.2). Therefore, the total value of items appearing only in edges in $B_{\leq \ell}$ is at most $(1/\beta)(|B_{\leq \ell}| - 1) < (1/\beta)(|P_{\leq \ell}|)$[4], by the minimality of β-edges (see Remark 3.4.1). Indeed, if an edge in B_ℓ has more than τ/β items not appearing in an edge in $A_{\leq \ell+1} \cup I$ then those items would form a thin β-edge which contradicts its minimality.

Using $|A_{\leq \ell+1} \cup I| < (\mu+\gamma)|P_{\leq \ell}|$ we have

$$\sum_{i\in\mathscr{P}} y_i - \sum_{j\in\mathscr{R}} z_j > (1-\gamma)\left(1-\frac{1}{\alpha}\right)|P_{\leq \ell}| - (\mu+\gamma)\left(\frac{1}{\alpha}+\frac{1}{\beta}\right)|P_{\leq \ell}| - \frac{1}{\beta}|P_{\leq \ell}|.$$

Recall that, given any feasible solution to the dual of C-LP(τ), we can scale it by any positive number, and it will remain feasible: This implies that if the optimum of the dual of C-LP(τ) is positive, then the dual of C-LP(τ) is unbounded. Hence, the dual of C-LP(τ) is unbounded when

$$(1-\gamma)\left(1-\frac{1}{\alpha}\right) - (\mu+\gamma)\left(\frac{1}{\alpha}+\frac{1}{\beta}\right) - \frac{1}{\beta} \geq 0 \Leftrightarrow \gamma \leq \frac{\alpha\beta - (1+\mu)(\alpha+\beta)}{\alpha\beta+\alpha}.$$

Recall that $\beta = 2(3+\sqrt{10})+\epsilon$, $\alpha = 2$, and $\mu = \epsilon/100$. For $\epsilon > 0$ the last inequality is equivalent to $206\sqrt{10} + 3\epsilon \leq 676$, which is valid for $\epsilon \leq 1$. □

We remark that the above lemma states the only condition that needs to be satisfied for the algorithm to run in polynomial time. Therefore, in a binary search over the possible values of τ, the algorithm can abort if the above condition is violated at any time, because that violation would imply that the configuration-LP is infeasible; otherwise it will terminate in polynomial time.

We now use the previous lemma to show that, if we create a new layer, then the number of agents in that layer will increase rapidly. This will enable us to bound the number of layers to be logarithmic and also to bound the running time.

Lemma 3.4.7 (Exponential growth)**.** *At each execution of the iterative step of the algorithm, we have*

$$|P_i| \geq \delta|P_{\leq i-1}|, \text{ where } \delta := \epsilon/100,$$

for each $i = 1, \ldots, \ell$.

Proof. Suppose towards contradiction that the statement is false and let t be the smallest index that violates it, i.e., $|P_t| < \delta|P_{\leq t-1}|$. Due to the definition of collapsible layers, $|I_i| < \mu|P_i|$

[4]Remember B_0 is conventionally set to $\{(p_0, \emptyset)\}$.

for $0 \leq i \leq t$. Hence,

$$|I_{\leq t}| < \mu |P_{\leq t}| < \mu(1+\delta)|P_{\leq t-1}|.$$

Further,

$$|A_{\leq t}| + |I_{\leq t}| \geq \mathrm{DP}_M(P_{\leq t-1}, A_{\leq t} \cup I_{\leq t}) = \mathrm{DP}_M(P_{\leq t-1}, A_{\leq t} \cup I) \geq \gamma |P_{\leq t-1}|,$$

where the first inequality is trivial, the equality follows from the definition of canonical decompositions (Definition 3.4.8), and the last inequality follows from Lemma 3.4.6. This gives us

$$|A_{\leq t}| > \big(\gamma - \mu(1+\delta)\big)\,|P_{\leq t-1}|.$$

We now obtain an upper bound on the total number of edges in $A_{\leq t}$ by counting the value of items in each A_i and B_i; observe that any thin β-edge has items of total value at most $2\tau/\beta$ due to minimality, whereas any thin α-edge in $A_{\leq t}$ has items of value at least $\tau/\alpha - \tau/\beta$ that are blocked, i.e., appear in some edge in $B_{\leq t}$ (since otherwise this edge would be in I instead of $A_{\leq t}$)[5]. Hence,

$$|A_i|\big(\tau/\alpha - \tau/\beta\big) \leq |B_i|\big(2\tau/\beta\big) \overset{\text{summing over } i \text{ and rearranging}}{\Longrightarrow} |A_{\leq t}| \leq |B_{\leq t}| \frac{2\alpha}{\beta - \alpha}.$$

Since $|B_{\leq t}| < |P_{\leq t}|$ and $|P_{\leq t}| < (1+\delta)|P_{\leq t-1}|$ we have the bound

$$|A_{\leq t}| < \frac{2\alpha}{\beta - \alpha}(1+\delta)|P_{\leq t-1}|.$$

Therefore we will have a contradiction when

$$\frac{2\alpha}{\beta - \alpha}(1+\delta) \leq \gamma - (1+\delta)\mu.$$

It can be verified that for any $\epsilon > 0$ the above inequality is equivalent to

$$22400 + 6\left(52 + \sqrt{10}\right)\epsilon + 3\epsilon^2 \leq 9400\sqrt{10},$$

which is true for $\epsilon \in [0,1]$ leading to the required contradiction. □

We are now ready to prove that our algorithm executes a polynomial number of iterations. To

[5] Observe that all edges in A_i have all but at most τ/β items blocked. Otherwise, the edge is added to I in the build phase; and if items have been freed up later, the edge is removed from A_i (and it can be added to I) during Step (ii) of the collapse phase.

do this, we define the signature vector $s := (s_0, \ldots, s_\ell, \infty)$, where

$$s_i := \lfloor \log_{1/(1-\mu)} \frac{|P_i|}{\delta^{i+1}} \rfloor$$

corresponding to the state $(M, p_0, \ell, \{(A_0, d_0), \ldots, (A_{\ell+1}, d_{\ell+1})\}, I)$ of the algorithm. The signature vector changes as the algorithm executes. In fact, we prove that its lexicographic value always decreases:

Lemma 3.4.8. *Across each iterative step, the lexicographic value of the signature vector decreases. Furthermore, the coordinates of the signature vector are always non-decreasing.*

Proof. We show this by induction as usual on the variable r that counts the number of times the iterative step has been executed. The statement for $r = 0$ is immediate. Suppose it is true for $r \geq 0$. Let $s = (s_0, \ldots, s_\ell, \infty)$ and $s' = (s'_0, \ldots, s'_{\ell'}, \infty)$ denote the signature vector at the beginning and at the end of the $(r+1)$-th iterative step. We consider two cases:

No layer was collapsed Let $L_{\ell+1}$ be the newly constructed layer. In this case, $\ell' = \ell + 1$. By Lemma 3.4.7, $|P_{\ell+1}| \geq \delta |P_{\leq \ell}| > \delta |P_\ell|$. Clearly, $s' = (s_0, \ldots, s_\ell, s'_{\ell+1}, \infty)$ where $\infty > s'_{\ell+1} \geq s'_\ell = s_\ell$. Thus, the signature vector s' also has increasing coordinates and smaller lexicographic value compared to s.

At least one layer was collapsed Let $0 \leq t \leq \ell$ be the index of the last layer that was collapsed during the r-th iterative step. As a result of the collapse operation suppose the layer P_t changed to P'_t. Then we know that $|P'_t| < (1-\mu)|P_t|$. Indeed, during the collapse phase of our combinatorial algorithm, at least a μ-fraction of the edges in B_t are replaced with edges from I. Since none of the layers with indices less than t were affected during this procedure, $s' = (s_0, \ldots, s_{t-1}, s'_t, \infty)$ where $s'_t = \lfloor \log_{1/(1-\mu)} \frac{|P'_t|}{\delta^{t+1}} \rfloor \leq \lfloor \log_{1/(1-\mu)} \frac{(1-\mu)|P_t|}{\delta^{t+1}} \rfloor \leq \lfloor \log_{1/(1-\mu)} \frac{|P_t|}{\delta^{t+1}} \rfloor - 1 = s_t - 1$. This shows that the lexicographic value of the signature vector decreases, and that the coordinates of s' are non-decreasing follows from Lemma 3.4.7. □

Finally, due to the above lemma, any upper bound on the number of possible signature vectors is an upper bound on the number of iterations our combinatorial algorithm will execute; We prove there is such a bound of polynomial size:

Lemma 3.4.9. *The number of signature vectors is at most* $|\mathscr{P}|^{O(1/\mu \cdot 1/\delta \cdot \log(1/\delta))}$.

Proof. By Lemma 3.4.7, $|\mathscr{P}| \geq P_{\leq \ell} \geq (1+\delta) P_{\leq \ell - 1} \geq \cdots \geq (1+\delta)^\ell |P_0|$. This implies that $\ell \leq \log_{1+\delta} |\mathscr{P}| \leq \frac{1}{\delta} \log |\mathscr{P}|$, where the last inequality is obtained by using Taylor series, and that $\delta \in [0, 1/100]$.

Now consider the i-th coordinate of the signature vector s_i. It can be no greater than $\log_{1/(1-\mu)} \frac{|\mathscr{P}|}{\delta^{i+1}}$. Using the bound on the index i and after some manipulations, we get

$$\begin{aligned} s_i &\leq \left(\log|\mathscr{P}| + (i+1)\log\frac{1}{\delta}\right)\frac{1}{\log\frac{1}{1-\mu}} \\ &\leq \left(\log|\mathscr{P}| + (\frac{1}{\delta}\log|\mathscr{P}| + 1)\log\frac{1}{\delta}\right)\frac{1}{\log\frac{1}{1-\mu}} \\ &= \log|\mathscr{P}| \cdot O\left(\frac{1}{\mu\delta}\log\frac{1}{\delta}\right), \end{aligned}$$

where the final bound is obtained by again expanding using Taylor series around 0. Thus, if we let $U = \log|\mathscr{P}| \cdot O\left(\frac{1}{\mu\delta}\log\frac{1}{\delta}\right)$ be an upper bound on the number of layers and the value of each coordinate of the signature vector, then the sum of coordinates of the signature vector is always upper bounded by U^2.

Here, as in the simpler algorithm, we apply the bound on the number of partitions of an integer. Recall that the number of partitions of an integer N can be upper bounded by $e^{O(\sqrt{N})}$ [20]. Since each signature vector corresponds to some partition of an integer at most U^2, we can upper bound the total number of signature vectors by $\sum_{i \leq U^2} e^{O(\sqrt{i})}$.

Using the bound of U, we have that the number of signatures is at most $|\mathscr{P}|^{O(1/\mu \cdot 1/\delta \cdot \log(1/\delta))}$. □

As the number of possible signature vectors is polynomial, the number of iterations our combinatorial algorithm will execute is also polynomial. Furthermore, as the running time of each iteration is also polynomial, this completes the proof of Lemma 3.4.1, hence of Theorem 3.1.1.

4 Scheduling Jobs with Uniform Smith Ratios

The results of this chapter are based on a joint work with Ola Svensson and Jakub Tarnawski; this work will be published in SODA 2017 [27].

4.1 Introduction

The problem we study is that of scheduling jobs on unrelated machines in order to minimize the weighted sum of completion times. Formally, we are given a set $\mathscr{M}$ of machines, and a set $\mathscr{J}$ of jobs, each with a weight $w_j \geq 0$, such that the processing time (also called *size*) of job j on machine i is $p_{ij} \geq 0$. The objective is to find a schedule that minimizes the weighted completion time, $\sum_{j \in \mathscr{J}} w_j C_j$, where C_j denotes the completion time of job j in the constructed schedule. In the three-field notation used in scheduling literature [17], this problem is denoted as $R||\sum w_j C_j$.

The weighted completion-time objective, along with makespan and flow-time minimization, is one of the most relevant and well-studied objectives for measuring the quality of service in scheduling. Already in 1956, Smith [42] showed a simple rule for minimizing this objective on a single machine: schedule the jobs in non-increasing order of w_j/p_j (where p_j denotes the processing time of job j on the single machine). This order is often referred to as the Smith ordering of the jobs and the ratio w_j/p_j is called the Smith ratio of job j. In the case of parallel machines, the problem becomes significantly harder. Already for identical machines (the processing time of a job is the same on all machines), it is strongly **NP**-hard, and for the more general unrelated machine model that we consider, the problem is **NP**-hard to approximate within $1+\varepsilon$, for a small $\varepsilon > 0$ [22].

Skutella and Woeginger [41] settled the approximability for identical machines by developing a polynomial time approximation scheme. That is, for every $\varepsilon > 0$, they gave a $(1+\varepsilon)$-approximation algorithm for minimizing the weighted sum of completion times on identical parallel machines.

In contrast, there remains a notorious open problem in scheduling theory about settling the ap-

proximability in the unrelated machine model (see e.g. "open problem 8" in [37]). First, Schulz and Skutella [36], and independently Chudak [12], came up with $(3/2+\epsilon)$-approximation algorithms, that employ a time-indexed LP relaxation for the problem. Shortly thereafter, the approximation guarantee was improved to $3/2$, by Skutella [40] and Sethuraman and Squillante [38], by using a clever convex quadratic programming relaxation. All these results relied on designing a convex relaxation and then on applying *independent* randomized rounding with the marginal probabilities that were returned by the convex relaxation solution. The analysis of these algorithms is in fact tight: it is not hard to see that any algorithm using independent randomized rounding cannot achieve a better approximation guarantee than $3/2$. Recently, Bansal et al. [5] overcame this barrier by designing a randomized rounding scheme that informally enhances independent randomized rounding by introducing strong negative correlation properties. Their techniques yield a $(3/2-\varepsilon)$-approximation algorithm with respect to either a semidefinite programming relaxation, introduced by themselves, or the configuration-LP relaxation introduced in [45]. Their rounding and analysis improve, and they build upon methods used previously for independent randomized rounding. So a natural question behind this work is, Can a different rounding approach yield significant improvements of the approximation guarantee?

4.1.1 Our Results

Departing from previous rounding approaches, we propose to use the same elegant rounding scheme for the weighted completion-time objective, as devised by Shmoys and Tardos [39] for optimizing a linear function subject to makespan constraints on unrelated machines. We give a *tight* analysis that shows that this approach gives a significantly improved approximation guarantee in the special case where the Smith ratios of all jobs that can be processed on a machine are uniform: that is, we have $p_{ij} \in \{w_j, \infty\}$ for all $i \in \mathscr{M}$ and $j \in \mathscr{J}$.[1]

This restriction, which has not been studied previously, captures the natural notion that any unit of work (processing time) on a fixed machine has the same weight. It corresponds to the class of instances where the order of jobs on a machine does not matter. Compared to another natural restriction of $R||\sum w_j C_j$ – namely, the *unweighted* sum of completion times $R||\sum C_j$ – it is both computationally harder and more intuitive. It is computationally harder because our problem inherits all the known hardness characteristics of the general weighted version (see Section 4.1.2), whereas the unweighted version is polynomial-time solvable [23, 8]; and it is more intuitive because it is reasonable to expect that larger jobs have larger significance. Despite the negative results, our main theorem indicates that we understand this version far better than we do the general case.

[1] This restriction could be seen as close to the restricted assignment problem. However, we remark that all our results also apply to the more general (but also more notation-heavy) case where the weight of a job can also depend on the machine. A general version of our assumption then becomes $p_{ij} \in \{\alpha_i w_{ij}, \infty\}$ for some machine-dependent $\alpha_i > 0$. Our results apply to this version because our analysis will be done locally for each machine i. Therefore, we will only require that the Smith ratios be uniform for each machine separately.

To emphasize that we are considering the case where the weight of a job is proportional to its processing time, we refer to this problem as $\boldsymbol{R||\sum p_j C_j}$ (with p_j as opposed to w_j). With this notation, our main result can be stated as follows:

Theorem 4.1.1. *For any $\varepsilon > 0$, there exists a $\frac{1+\sqrt{2}}{2} + \varepsilon < 1.21$-approximation algorithm for $R||\sum p_j C_j$. Moreover, the analysis is tight: there exists an instance for which our algorithm returns a schedule with objective value at least $\frac{1+\sqrt{2}}{2} - \varepsilon$ times the optimum value.*

We remark that the ε in the approximation guarantee arises because we can only solve the configuration-LP relaxation (see Section 4.2) up to any desired accuracy.

Interestingly enough, a similar problem (namely, scheduling jobs with uniform Smith ratios on *identical parallel machines*) was studied by Kawaguchi and Kyan [28]. They achieve, by using a greedy list-scheduling algorithm, the same approximation ratio as we do. In their analysis, they reduce the problem of upper-bounding the output cost to that of estimating the output cost of certain worst-case instances. Then, in order to analyze the output cost of these instances, they use ideas similar to those we use for analyzing the output cost of the worst-case instances of our algorithm.

As we use the rounding algorithm by Shmoys and Tardos, a pleasant side-effect is that our algorithm can also serve as a bi-criteria $(1+\sqrt{2})/2+\varepsilon$-approximation for the $\sum p_j C_j$ objective and 2-approximation for the makespan objective:

Theorem 4.1.2. *For $R||\sum p_j C_j$, there exists an algorithm that, given any makespan threshold $T > 0$, returns a schedule with makespan at most $2T + \varepsilon$ and cost (i.e., sum of weighted completion times) within a factor $(1+\sqrt{2})/2+\varepsilon$ of the lowest-cost a schedule among those with makespan at most T can achieve.*

Again, we remark that the ε in the cost and makespan guarantees arises because we can only solve the corresponding configuration-LP relaxation (see Section 4.6) up to any desired accuracy. This bi-objective setting was previously studied by Kumar et al. [30], who gave a bi-criteria 3/2-approximation for the general weighted completion-time objective and 2-approximation for the makespan objective.

Our main technical contribution is a tight analysis of the algorithm, with respect to the strong configuration-LP relaxation. Configuration-LPs have been used to design approximation algorithms for multiple important allocation problems, often with great success. Therefore, as first noted by Sviridenko and Wiese [45], they constitute a promising direction to explore in search for better algorithms for $R||\sum w_j C_j$. We hope that our analysis can give further insights as to how the configuration-LP can be used to achieve this.

On a high level, our analysis proceeds as follows. A fractional solution to the configuration-LP defines, for each machine, a local probability distribution of the set of jobs (configuration) that will be processed by that machine. At the same time, the rounding algorithm (naturally)

produces a global distribution over such assignments, that inherits certain constraints on the local distribution for each machine. Therefore, focusing on a single machine, we will compare the local input distribution (i.e., the one defined by the configuration-LP) to the *worst possible* (among those satisfying said constraints) local output-distribution that could be returned by our randomized rounding.[2] In order to analyze this ratio, we will put both distributions under a series of transformations that can only worsen the guarantee, until we bring them into such a form that computing the exact approximation ratio is possible. As the final form also naturally corresponds to a scheduling instance, the tightness of our analysis follows immediately.

4.1.2 Lower Bounds and Hardness

All the known hardness features of the general problem $R||\sum w_jC_j$ transfer to our version $R||\sum p_jC_j$.

First, APX-hardness for $R||\sum w_jC_j$ was first proved by Hoogeveen et al. [22]. Skutella [40, Section 7] gives a different proof, where the reduction generates instances with all jobs having weights equal to processing times. Hence, APX-hardness for $R||\sum p_jC_j$ is established as well.

Furthermore, complementing the $(1+\sqrt{2})/2$ upper bound on the integrality gap of the configuration-LP that follows from our algorithm, we have the following lower bound, proved in Section 4.5:

Theorem 4.1.3. *The integrality gap of the configuration-LP for* $R||\sum p_jC_j$ *is at least* $13/12$.

To the best of our knowledge, this is also the best-known lower bound on the integrality gap of the configuration-LP for $R||\sum w_jC_j$.

Finally, recall that the $\frac{3}{2}$-approximation algorithms for the general problem $R||\sum_j w_jC_j$, by Skutella [40] and by Sethuraman and Squillante [38], are based on an *independent* randomized rounding of a fractional solution to a convex programming relaxation. It was shown by Bansal et al. [5] that this relaxation has an integrality gap of $\frac{3}{2}$ and that no independent randomized rounding algorithm can have an approximation ratio better than $\frac{3}{2}$. We note that both of these claims also apply to our version. Indeed, the integrality gap example of Bansal et al. can be modified to hold for $R||\sum_j p_jC_j$ (we discuss this integrality gap in Section 4.2). For the second claim, their problematic instance already has only unit sizes and weights. Thus, to achieve better than a $\frac{3}{2}$-approximation for our version, we cannot use independent randomized rounding or the relaxation of [40, 38].

[2] For example, if we consider all distributions that assign 2 jobs to a machine, each with probability $1/2$, then the distribution that assigns either both jobs together or no job at all, each with probability $1/2$, is the worst possible distribution, i.e., the one that maximizes the expected cost.

4.1.3 Chapter Overview

This chapter is structured as follows. In Section 4.2, we start by introducing some notation and by restating the configuration-LP. Then, in Section 4.3, we describe the randomized rounding algorithm, by Shmoys and Tardos [39], applied to our setting. We analyze it Section 4.4. Finally, we present the proof of the lower bound on the integrality gap in Section 4.5, and the proof of Theorem 4.1.2 in Section 4.6.

4.2 Preliminaries

Recall that $p_{ij} \in \{w_j, \infty\}$ for all $i \in \mathcal{M}$ and $j \in \mathcal{J}$, and that we let $\mathcal{J}_i = \{j \in \mathcal{J} : p_{ij} = p_j\}$ denote the set of jobs that can be assigned to machine $i \in \mathcal{M}$. Given $C_i \subseteq \mathcal{J}_i$, then we know (as we saw in the previous section) that scheduling the jobs in C_i on machine i by using the Smith ordering is optimal. As the Smith ratios of all jobs are the same, then scheduling C_i in machine i in any order achieves minimum cost. Hence, we have that

$$\text{cost}(C_i) = \sum_{j \in C_i} p_j^2 + \sum_{j \neq j' \in C_i} \frac{p_j p_{j'}}{2}.$$

where $\text{cost}(C_i)$ is the cost of scheduling jobs C_i on machine i. To see this, note that if we pick a random schedule/permutation of C_i, then the expected completion time of job j is $p_j + \sum_{j' \neq j \in C_i} \frac{p_{j'}}{2}$, and recall that the weight of j is $w_j = p_j$. The total cost of the considered schedule is $\sum_{i \in \mathcal{M}} \text{cost}(C_i)$.

Now, remember the configuration-LP for $R||\sum p_j C_j$:

$$\begin{aligned}
\min \quad & \sum_{i \in \mathcal{M}} \sum_{C \subseteq \mathcal{J}_i} y_{iC} \, \text{cost}(C) \\
\text{s.t.} \quad & \sum_{C \subseteq \mathcal{J}_i} y_{iC} \leq 1 && \forall i \in \mathcal{M}, \\
& \sum_{i \in \mathcal{M}} \sum_{C \subseteq \mathcal{J}_i : j \in C} y_{iC} = 1 && \forall j \in \mathcal{J}, \\
& y_{iC} \geq 0 && \forall i \in \mathcal{M}, \, C \subseteq \mathcal{J}_i.
\end{aligned}$$

Observe that one way to interpret this LP relaxation is to view it as, for each machine, defining a local distribution over assignments of sets of jobs.

Comparison to Natural Convex Relaxation Now, let us provide some intuition about why using the configuration-LP is a promising approach to designing a good approximation algorithm for $R||\sum p_j C_j$. We do this by describing a more natural quadratic programming relaxation for $R||\sum p_j C_j$, and by showing how the configuration-LP has significantly better performance on the integrality gap instances of the natural relaxation. Let us first state the convex relaxation that was used by Skutella [40], and Sethuraman and Squillante [38] (although

originally used for $R||\sum w_jC_j$, we display the convex program modified for $R||\sum p_jC_j$). The relaxation contains a variable x_{ij} for each $i \in \mathscr{M}$ and $j \in \mathscr{J}$; in an integral solution, x_{ij} would be set to 1 if job j was assigned to machine i, and 0 otherwise. Notice that in any feasible solution of an instance of $R||\sum p_jC_j$, every job is assigned to exactly one machine. Therefore, any convex relaxation of $R||\sum p_jC_j$ should naturally ensure that, in an integral solution, for each job j only one variable x_{ij} is set to 1. The relaxation can be described as follows:

$$\begin{array}{rlll} \min & z & & \\ \text{subject to} & z & \geq c^T x & \\ & z & \geq \frac{1}{2}c^T x + \frac{1}{2}x^T D x & \\ & \sum_{i\in\mathscr{M}} x_{ij} & = 1 & \forall j \in \mathscr{J} \\ & x_{ij} & \geq 0 & \forall i \in \mathscr{M}, j \in \mathscr{J} \end{array}$$

where $c^T x = \sum_{i\in\mathscr{M}} \sum_{j\in\mathscr{J}} p_{ij}^2 x_{ij}$ and $x^T D x = \sum_{i\in\mathscr{M}} \sum_{j\in\mathscr{J}} x_{ij} p_{ij} (\sum_{j'\in\mathscr{J}:j\neq j'} p_{ij'} x_{ij'} + x_{ij} p_{ij})$. The relaxation includes two individual constraints, and one set of constraints (one constraint for each $j \in \mathscr{J}$): the first constraint lower bounds the cost by $c^T x$, the second constraint lower bounds the cost by $\frac{1}{2}c^T x + \frac{1}{2}x^T D x$, and the set of constraints states that every job should be assigned exactly once.

Let us now discuss why the above quadratic program (QP) is a convex QP-relaxation for the $R||\sum p_jC_j$ problem. To begin with, the third set of constraints guarantees that every integral solution to the QP is a valid assignment of jobs to machines. Furthermore, given an integral assignment x, its cost is

$$\sum_{i\in\mathscr{M}} \sum_{j\in\mathscr{J}} x_{ij} p_j (\sum_{j'\in\mathscr{J}:j'\neq j} x_{ij'} p_{ij'}/2 + x_{ij} p_{ij}) = \frac{1}{2}(c^T x + x^T D x) \geq c^T x$$

where we crucially used that $x_{ij}^2 = x_{ij}$ for $x_{ij} \in \{0,1\}$ (in general, $c^T x$ and $x^T D x$ are incomparable). Finally, it can be proved (e.g. in [40]) that D is positive semi-definite, and hence $c^T x + x^T D x$ is convex. Therefore, the QP we described is indeed a convex relaxation of $R||\sum p_jC_j$.

Let us now describe an instance of $R||\sum p_jC_j$ for which the integrality gap of the QP-relaxation is 3/2; this instance is a modification of the instance provided by Bansal et al. (see [5, Claim 2.1]) for $R||\sum w_jC_j$. The instance contains $k+1$ jobs $\mathscr{J} = \{j_1, \ldots, j_{k+1}\}$ and $k+1$ machines $\mathscr{M} = \{i_1, \ldots, i_{k+1}\}$. Jobs $\{j_1, \ldots, j_k\}$ have size 1 and can be assigned to machine i_1, whereas job j_{k+1} has size k and can be assigned to machines $\{i_2, \ldots, i_{k+1}\}$.

Now, consider the fractional solution to the QP-relaxation that assigns $x_{ij_{k+1}} = 1/k$ for all $i \in \{i_2, \ldots i_{k+1}\}$ (clearly all the other jobs have to be integrally assigned to machine i_1).

For this fractional solution, we have $c^T x = x^T D x = k^2 + k$, therefore $z \geq k^2 + k$, whereas any integral solution has cost at least $\frac{3}{2}k^2$. Furthermore, it is most interesting that the first job contributes k^2 to $c^T x$ and k to $x^T D x$, whereas the rest of the jobs contribute k to $c^T x$ and

k^2 to $x^T Dx$. Therefore, the two lower bounds on the cost ($c^T x$ and $x^T Dx$) perform badly for two different types of solutions: $c^T x$ performs badly when most of the cost of the fractional solution comes from machines that are assigned many (relatively small) jobs, while $x^T Dx$ performs badly when the fractional assignment of jobs is spread out over many machines.

The configuration-LP does not display these disadvantages: given an assignment of a set of jobs to a machine, the configuration-LP will estimate its cost accurately. Indeed, we can see that the configuration-LP has integrality gap of 1 for the above instance. Therefore, the only reason the configuration-LP might underestimate the optimal cost of an instance is it might output a set of local distributions over assignments of jobs to machines, and these distributions are not consistent, i.e., such that there exists no global distribution that produces the same local distributions for each machine.

4.3 The Rounding Algorithm

Here, we describe our approximation algorithm. We will analyze it in the next section, yielding Theorem 4.1.1. The first step of our algorithm is to solve the configuration-LP (approximately) to obtain a fractional solution $y^\star$. We then round this solution in order to retrieve an integral assignment of jobs to machines. The rounding algorithm that we employ is the same as that used by Shmoys and Tardos [39] (albeit applied to a fractional solution to the configuration-LP, instead of the so-called assignment-LP). We describe the rounding scheme in Algorithm 4.1; see also Figure 4.1.

The first step is to define $x_{ij} = \sum_{C \subseteq \mathscr{J}_i : j \in C} y^\star_{iC}$. Intuitively, x_{ij} denotes the marginal probability that job j should be assigned to machine i, according to $y^\star$. Note that, by the constraint that $y^\star$ assigns each job once (fractionally), we have $\sum_{i \in \mathscr{M}} x_{ij} = 1$ for each job $j \in \mathscr{J}$.

In the next steps, we round the fractional solution randomly so as to satisfy these marginals, i.e., so that the probability that job j is assigned to i is x_{ij}. In addition, the number of jobs assigned to a machine i will closely match the expectation $\sum_{j \in \mathscr{J}} x_{ij}$: our rounding will assign either $\lfloor \sum_{j \in \mathscr{J}} x_{ij} \rfloor$ or $\lceil \sum_{j \in \mathscr{J}} x_{ij} \rceil$ jobs to machine i. This is enforced by creating $\lceil \sum_{j \in \mathscr{J}} x_{ij} \rceil$ "buckets" for each machine i, and then matching the jobs to these buckets. More formally, this is modeled by the complete bipartite graph $G = (U \cup V, E)$ constructed in Step 2 of Algorithm 4.1, where vertex $u_{i,t} \in U$ corresponds to the t-th bucket of machine i.

Observe that any integral matching in G that matches all the "job" vertices in V naturally corresponds to an assignment of jobs to machines. Step 3 prescribes a distribution on such matchings by defining a fractional matching z. The procedure is as follows: For each machine i, we iterate over the jobs $j \in \mathscr{J}$ in *non-increasing order* in terms of their size, and we insert items of size x_{ij} into the first bucket until adding the next item would cause the bucket to become full. Then, we split that item between the first bucket and the second, and we proceed likewise for all jobs until we fill up all buckets (except possibly for the last bucket). Having completed this process for all machines i, we end up with a fractional matching z in G with

Input : Solution $y^\star$ to the configuration-LP
Output: Assignment of jobs to machines

1) Define $x \in \mathbb{R}^{\mathscr{M} \times \mathscr{J}}$ as follows: $x_{ij} = \sum_{C \subseteq \mathscr{J}_i : j \in C} y^\star_{iC}$.

2) Let $G = (U \cup V, E)$ be the complete bipartite graph where

- the right-hand side consists of one vertex for each job j, i.e., $V = \{v_j : j \in \mathscr{J}\}$,
- the left-hand side consists of $\lceil \sum_{j \in \mathscr{J}} x_{ij} \rceil$ vertices for each machine i, i.e., $U = \bigcup_{i \in \mathscr{M}} \{u_{i,t} : 1 \leq t \leq \lceil \sum_{j \in \mathscr{J}} x_{ij} \rceil\}$.

3) Define a fractional solution z to the bipartite matching LP for G (initially set to $z = \mathbf{0}$) by repeating the following procedure for every machine $i \in \mathscr{M}$:

- Let $k = \lceil \sum_{j \in \mathscr{J}} x_{ij} \rceil$, and let t be a variable originally set to 1.
- Iterate over all $j \in \mathscr{J}$ in *non-increasing order* in terms of p_j:

  If $x_{ij} + \sum_{j' \in \mathscr{J}} z_{u_{i,t} v_{j'}} \leq 1$, then set $z_{u_{i,t} v_j} = x_{ij}$.

  Else, set $z_{u_{i,t} v_j} = 1 - \sum_{j' \in \mathscr{J}} z_{u_{i,t} v_{j'}}$, increment t, and set $z_{u_{i,t} v_j} = x_{ij} - z_{u_{i,t-1} v_j}$.

4) Decompose z into a convex combination of integral matchings $z = \sum_t \lambda_t z_t$ and sample one integral matching z^* by choosing the matching z_t with probability λ_t.

5) Schedule $j \in \mathscr{J}$ on $i \in \mathscr{M}$ iff $z^*_{u_{i,t} v_j} = 1$ for some $1 \leq t \leq \lceil \sum_{j \in \mathscr{J}} x_{ij} \rceil$.

Algorithm 4.1: Randomized rounding

the following properties:

- Every "job" vertex $v_j \in V$ is fully matched in z, i.e., $\sum_{i,t} z_{u_{i,t} v_j} = 1$.
- For every "bucket" vertex $u_{i,t} \in U$, we have $\sum_j z_{u_{i,t} v_j} \leq 1$, with equality if $t < \lceil \sum_j x_{ij} \rceil$.
- The fractional matching preserves the marginals, i.e., $x_{ij} = \sum_t z_{u_{i,t} v_j}$ for all $j \in \mathscr{J}$ and $i \in \mathscr{M}$.
- We have the following bucket structure: if $z_{u_{i,t} v_j} > 0$ and $z_{u_{i,t'} v_{j'}} > 0$ with $t' > t$, then $p_j \geq p_{j'}$.

The last property follows because Step 3 considered the jobs in non-increasing order of their processing times. This will be important in the analysis (see the last property of Fact 4.4.1).

Here, we want to randomly select a matching for G, which satisfies the marginals of z (remember that such a matching corresponds to an assignment of all the jobs to machines). We know that the bipartite matching LP is integral and that z is a feasible solution for the bipartite matching LP of G. Therefore, by using an algorithmic version of Carathéodory's theorem (see e.g. Theorem 6.5.11 in [18]), we can decompose z into a convex combination $z = \sum_t \lambda_t z_t$ of

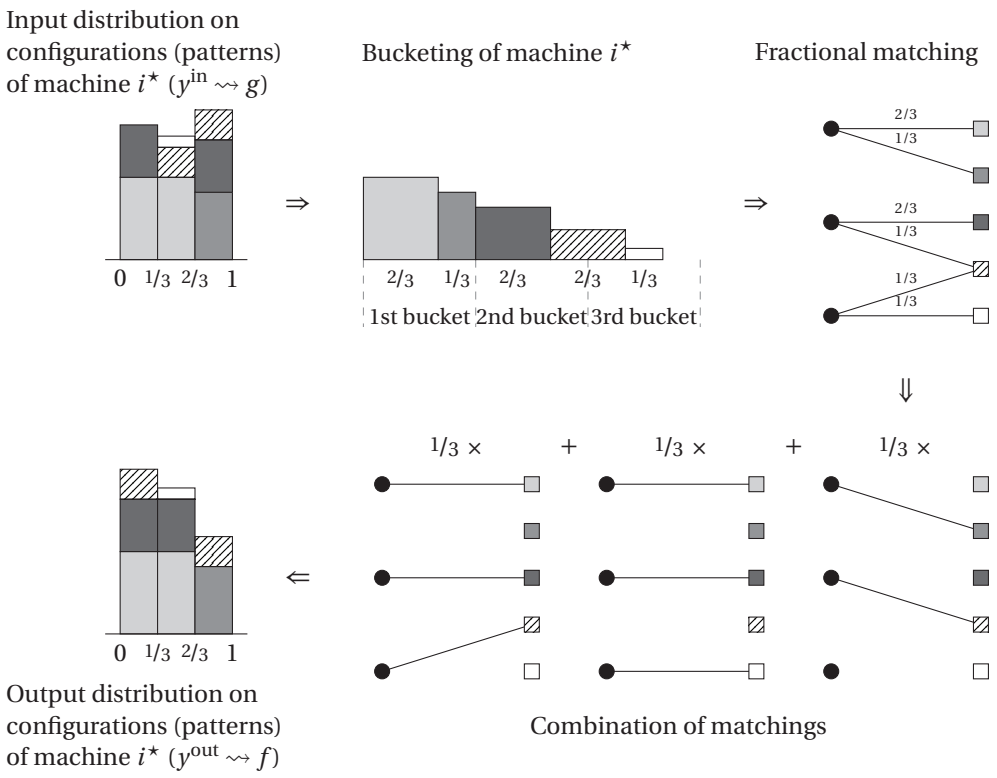

Figure 4.1: A sample execution of our rounding algorithm, restricted to a single machine $i^\star$. Jobs are represented by a rectangle; its height is the job's processing time and its width is its fractional assignment to $i^\star$. Starting from an input distribution over configurations for $i^\star$, we extract the fractional assignment of each job to $i^\star$, we create a bipartite graph consisting of 3 copies of $i^\star$ and the jobs that are fractionally assigned to it, and then we connect the jobs to the copies of $i^\star$ by iterating through the jobs in non-increasing order of p_j. Finally, we decompose the resulting fractional matching into a convex combination of integral matchings and we sample one of them. The shown output distribution is a worst-case distribution in the sense of Section 4.4.2: it maximizes the variance of makespan, subject to the marginal probabilities and the bucket structure enforced by the algorithm.

polynomially many integral matchings, and sample the matching z_t with probability λ_t. Then, if z^* is the matching we have sampled, we simply assign job j to machine i iff $z^*_{u_{i,t}v_j} = 1$ for some t. As $x_{ij} = \sum_t z_{u_{i,t}v_j}$ and $\sum_{i \in \mathcal{M}} x_{ij} = 1$ for all jobs j, z^* will match all "job" vertices.[3] The above steps are described in Steps 4 and 5 of Algorithm 4.1.

The entire rounding algorithm is depicted in Figure 4.1.

[3]We remark that this is the only part of the algorithm that employs randomness; in fact, we can derandomize the algorithm by choosing the matching z_k that minimizes the cost of the resulting assignment.

4.4 Analysis of the Rounding Algorithm

Throughout the analysis, we fix a single machine $i^\star \in \mathscr{M}$. We will show that the expected cost of our algorithm on this machine is at most $\frac{1+\sqrt{2}}{2}$ times the cost of the configuration-LP on this machine. This clearly implies (by linearity of expectation) that the expected cost of the produced solution (on all machines) is at most $\frac{1+\sqrt{2}}{2}$ times the cost of the LP solution (which is in turn within a factor $(1+\varepsilon)$ of the fractional optimum).

Let $C_1, C_2, ..., C_{2^{\mathscr{J}}}$ be all possible configurations sorted by decreasing cost, i.e., $\text{cost}(C_1) \geq \ldots \geq \text{cost}(C_{2^{\mathscr{J}}})$. To simplify notation, in this section we let $\mathscr{J}$ denote the set of jobs that can be processed on machine $i^\star$ (i.e., $\mathscr{J}_{i^\star}$).

Recall that the solution $y^\star$ to the configuration-LP gives us an *input* distribution on configurations assigned to machine $i^\star$, i.e., it gives us a vector $y^{\text{in}} \in [0,1]^{2^{\mathscr{J}}}$ such that $\sum_i y_i^{\text{in}} = 1$. With this notation, we can write the cost of the configuration-LP on machine $i^\star$ as

$$\sum_i y_i^{\text{in}} \text{cost}(C_i).$$

In order to compare this expression with the expected cost of our algorithm on machine $i^\star$, we observe that our rounding algorithm also gives a distribution on configurations. We denote this *output* distribution by $y^{\text{out}} \in [0,1]^{2^{\mathscr{J}}}$ (where $\sum_i y_i^{\text{out}} = 1$). Hence, the expected cost of our algorithm on machine $i^\star$ is

$$\sum_i y_i^{\text{out}} \text{cost}(C_i).$$

The result of this section, that which the approximation guarantee of Theorem 4.1.1, can now be stated as follows.

Theorem 4.4.1. *We have*

$$\frac{\sum_i y_i^{\text{out}} \, cost(C_i)}{\sum_i y_i^{\text{in}} \, cost(C_i)} \leq \frac{1+\sqrt{2}}{2}.$$

Our strategy for bounding this ratio is, broadly, to work on this pair of distributions by transforming it to another pair of distributions of special form, whose ratio we will be able to bound. We transform the pair in such a way that the ratio can only increase. In other words, we prove that no pair of distributions has a ratio worse than a certain worst-case kind of pair, and we bound the ratio in this worst case.

After these transformations, our pair of distributions might no longer correspond to the original scheduling problem instance, hence it will be convenient for us to work with a more abstract notion that we define now.

4.4.1 Compatible Function Pairs

Given a distribution $y \in [0,1]^{2^{\mathcal{J}}}$ with $\sum_i y_i = 1$, we can build a corresponding function f from $[0,1)$ to multisets of positive numbers as follows: define $f(x)$ for $x \in [0, y_1)$ to be the multiset of processing times of jobs in C_1, $f(x)$ for $x \in [y_1, y_1 + y_2)$ to be the multiset of processing times of jobs in C_2, and so on.[4] If we do this for both y^{out} – obtaining a function f – and y^{in} – obtaining a function g (see Figure 4.1 for an illustration of f and g), we will have produced a *function pair*:

Definition 4.4.1. *A* function pair *is a pair (f,g) of stepwise-constant functions from the interval $[0,1)$ to multisets of positive numbers. We will call these multisets* patterns *and the numbers they contain* elements *(or* processing times*).*

Notation. If f is such a function, define:

- $f_1 : [0,1) \to \mathbb{R}_+$ as the maximum element: $f_1(x) = \max f(x)$ (set 0 if $f(x) = \emptyset$),
- $\text{size}_f : [0,1) \to \mathbb{R}_+$ as $\text{size}_f(x) = \text{size}(f(x))$, where

$$\text{size}(f(x)) = \sum_{p \in f(x)} p,$$

- f_r as the total size of the multiset after the removal of the maximum: $f_r(x) = \text{size}_f(x) - f_1(x)$,
- $\text{cost}(f) = \int_0^1 \text{cost}(f(x))\,dx$ as the fractional (expected) cost, where

$$\text{cost}(f(x)) = \sum_{p \in f(x)} p \cdot \left(\sum_{q \in f(x), q \preceq p} q \right)$$

  for an arbitrary linear order $\preceq$ on $f(x)$.[5]

The function pairs we work with will have special properties that follow from the algorithm. We argue about them in Fact 4.4.1. One such property comes from our algorithm preserving the marginal probabilities of jobs:

Definition 4.4.2. *We say that a function pair (f,g) is a* compatible function pair (CFP) *if the fractional number of occurences of any element is the same in f and in g.*[6]

Fact 4.4.1. *Let (f,g) be a function pair obtained from $(y^{\text{out}}, y^{\text{in}})$ as described above. Then:*

[4] Recall that $C_1, C_2, \ldots$ are sorted by non-increasing cost. Thus, f can be thought of as a quantile function (inverse cumulative distribution function) of the distribution y, except in reverse order (i.e., $f(1-x)$ is a quantile function).

[5] This expression does not depend on $\preceq$, as the Smith ratios are uniform, and it is equal to the cost of a configuration giving rise to $f(x)$.

[6] Formally, for each $p > 0$ we have: $\int_0^1$ multiplicity of p in $f(x)\,dx = \int_0^1$ multiplicity of p in $g(x)\,dx$.

- (f, g) *is a CFP.*
- $cost(f) = \sum_i y_i^{\text{out}} cost(C_i)$ *and* $cost(g) = \sum_i y_i^{\text{in}} cost(C_i)$.
- f *has the following* bucket structure*: for any two patterns P and Q in the image of f and for any i, the i-th largest element of P is no smaller than the $(i+1)$-th largest element of Q.*

Proof. That (f, g) is a CFP follows because our algorithm satisfies the marginals of the involved jobs. Specifically, we know that for each job $j \in \mathcal{J}$, both distributions y^{in} and y^{out} have the machine $i^\star$ process a fraction $x_{i^\star j} = \sum_{C \ni j} y^\star_{i^\star C}$ of this job (where $y^\star$ is the global configuration-LP solution). For any $p > 0$, summing this up over all jobs $j \in \mathcal{J}$ with processing time $p_j = p$ gives the compatibility condition.

The equalities of costs are clear from the definition of cost(f) and cost(g).

For the bucket structure of f, recall the algorithm: A matching is found between jobs and buckets, in which each bucket is matched with a job (except potentially for the last bucket of each machine). For any pattern P in the image of f and for any i, the i-th processing time in P is drawn from the i-th bucket that was constructed by our algorithm. Moreover, all processing times in the i-th bucket are no smaller than those in the $(i+1)$-th bucket, because the algorithm orders jobs non-increasingly by processing times. (See Figure 4.1 for an illustration of this process and of a function f satisfying this bucket structure.) □

This was the last point in the analysis where we reasoned about how the algorithm rounds the LP solution. Henceforth, we will think about elements, patterns and CFPs rather than jobs and configurations.

To prove Theorem 4.4.1, we need to show that $\frac{\text{cost}(f)}{\text{cost}(g)} \leq \frac{1+\sqrt{2}}{2}$. As indicated above, we will do this by proving that there is another CFP (f', g') with special properties and such that $\frac{\text{cost}(f)}{\text{cost}(g)} \leq \frac{\text{cost}(f')}{\text{cost}(g')}$. We will actually construct a series of such CFPs in a series of lemmas, obtaining more and more desirable properties, until we can bound the ratio. Our final objective is a CFP like the pair (f', g') depicted in Figure 4.3.

4.4.2 The Worst-Case Output

As a first step, we look at how costly an output distribution of our algorithm can be (while still satisfying the aforementioned bucket structure and the marginal probabilities, i.e., the compatibility condition). Intuitively, the maximum-cost f is going to maximize the variance of the total processing time, which means that larger-size patterns should select larger processing times from each bucket. (See Figure 4.1 for an illustration and the proof of Lemma 4.4.1 for details.) From this, we extract that the largest processing time in a pattern (the function f_1) should be non-increasing, and this should also hold for the second-largest processing time, the third-largest, and so on. This implies the following properties:

Lemma 4.4.1. *If (f,g) is a CFP where f has the bucket structure described in Fact 4.4.1, then there exists another CFP (f',g) such that $\frac{cost(f)}{cost(g)} \leq \frac{cost(f')}{cost(g)}$ and the functions f'_1 and f'_r are non-increasing and size$(f'(1)) \geq f'_r(0)$.*

The proof is a simple swapping argument.

Proof. For $i = 1,2,...$, let $f_i(x)$ always denote the i-th largest element of $f(x)$. As suggested above, we will make sure that for each i, the function f_i is non-increasing.

Specifically, we repeat the following procedure: as long as there exist x, y and i such that $\text{size}(f(x)) > \text{size}(f(y))$ but $f_i(x) < f_i(y)$, swap the i-th largest elements in $f(x)$ and $f(y)$.[7]

Once this is no longer possible, we finish by "sorting" f so as to make size_f non-increasing.

Let us verify that once this routine finishes, yielding the function f', we have the desired properties:

- The function f'_1 is non-increasing.
- The same holds for the function f'_r, since $f'_r = f'_2 + f'_3 + ...$ and each f'_i is non-increasing.
- The procedure maintains the bucket structure, since we only swap patterns that belong to the same bucket. This implies that $f'_i(x) \geq f'_{i+1}(y)$ for all i, x and y. Thus

$$\text{size}(f'(1)) = f'_1(1) + f'_2(1) + ... \geq f'_2(0) + f'_3(0) + ... = f'_r(0).$$

- It remains to show that $\text{cost}(f') \geq \text{cost}(f)$. Without loss of generality, assume there was only a single swap (as the sorting step is insignificant for the cost). For computing the cost of the involved patterns, we suppose that the involved elements went last (as the order does not matter); let $R_x = \text{size}(f(x)) - f_i(x)$ and $R_y = \text{size}(f(y)) - f_i(y)$ be the total sizes of the elements not involved. Then $R_x > R_y$, since x, y and i were chosen such that $\text{size}(f(x)) > \text{size}(f(y))$ and $f_i(x) < f_i(y)$, and we have

$$\begin{aligned}
\Delta\,\text{cost}(f(x)) &= f_i(y)\left(R_x + f_i(y)\right) - f_i(x)\left(R_x + f_i(x)\right) \\
&= \left(f_i(y) - f_i(x)\right)R_x + f_i(y)^2 - f_i(x)^2, \\
\Delta\,\text{cost}(f(y)) &= f_i(x)\left(R_y + f_i(x)\right) - f_i(y)\left(R_y + f_i(y)\right) \\
&= \left(f_i(x) - f_i(y)\right)R_y + f_i(x)^2 - f_i(y)^2,
\end{aligned}$$

  thus

$$\Delta\,\text{cost}(f(x)) + \Delta\,\text{cost}(f(y)) = \left(f_i(y) - f_i(x)\right)\left(R_x - R_y\right) > 0.$$

[7] Formally, choose $\tau > 0$ such that f is constant on $[x, x+\tau)$ and on $[y, y+\tau)$ and perform the swap in these patterns.

□

4.4.3 Liquification

One of the most important operations we employ is called *liquification*. It is the process of replacing an element (processing time) with many tiny elements of the same total size. These new elements all have a size of ε and are called *liquid elements*. Elements of a size larger than ε are called *solid*. We should think that ε is arbitrarily small, much smaller than any p_j, hence we will usually work in the limit $\varepsilon \to 0$.

The intuition behind applying this process to our pair is that in the ideal worst-case setting which we are moving towards, there are only elements of two sizes: large and infinitesimally small. We will keep a certain subset of the elements intact (to play the role of large elements), and liquify the rest in Lemma 4.4.2.

Our main claim in this section is that replacing an element with smaller ones of the same total size (in both f and g) can only increase the ratio of costs. Hence, we are free to liquify elements in our analysis, as long as we make sure to liquify the same amount of every element in both f and g (f and g remain compatible).

Fact 4.4.2. *Let (f,g) be a CFP and $p, p_1, p_2 > 0$ with $p = p_1 + p_2$. Suppose (f',g') is a CFP obtained from (f,g) by replacing p by p_1 and p_2 in subsets of patterns in f and g of equal measures.*[8] *Then $\frac{cost(f)}{cost(g)} \le \frac{cost(f')}{cost(g')}$.*

Proof. Consider a pattern P in which p was replaced. We calculate the change in cost $\Delta := \text{cost}(P \setminus \{p\} \cup \{p_1, p_2\}) - \text{cost}(P)$. As the order does not matter, the cost can be analyzed with p (or p_1, p_2) being first, so

$$\Delta = (p_1^2 + p_2(p_1 + p_2)) - p^2 = (p_1^2 + p_1 p_2 + p_2^2) - (p_1 + p_2)^2 = -p_1 p_2 \le 0$$

and it does not depend on the other elements in P. Thus, if we make the replacement in a fraction τ of patterns, then we have

$$\frac{\text{cost}(f')}{\text{cost}(g')} = \frac{\text{cost}(f) + \tau\Delta}{\text{cost}(g) + \tau\Delta} \ge \frac{\text{cost}(f)}{\text{cost}(g)}$$

since $\frac{\text{cost}(f)}{\text{cost}(g)} \ge 1$ to begin with (otherwise we are done) and $\Delta \le 0$. □

By corollary, we can also replace an element of size p with p/ε liquid elements (of size ε each).

[8] Formally, suppose $I_f, I_g \subseteq [0,1)$ are finite unions of disjoint intervals of equal total length such that all patterns $f(x)$ and $g(y)$ for $x \in I_f$, $y \in I_g$ contain p; in all these patterns, remove p and add p_1, p_2.

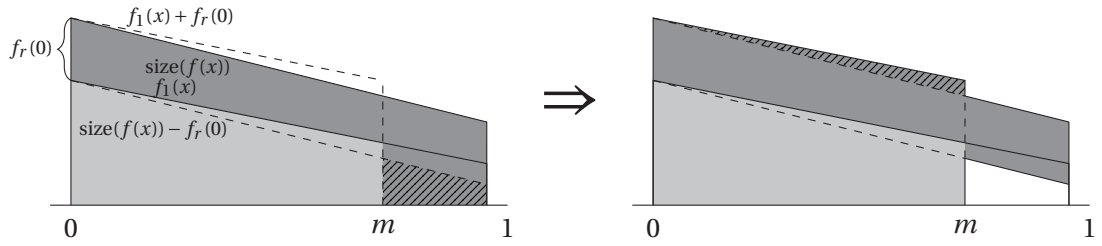

Figure 4.2: The main step in the proof of Lemma 4.4.2. The left picture shows f after the liquification of all elements except one largest element (f_1) for $x \in [0, m)$. The right picture shows f after the movement of liquid elements between the striped regions. In both pictures, the light-gray areas contain single large elements, and the dark-gray areas are liquid elements. Note that there are two pairs of parallel lines in the pictures; the vertical distance between each pair is $f_r(0)$. The threshold m is defined so that the striped regions have equal areas (thus f'_r is made constant by the movement of liquid elements) and so that moving the liquid elements can only increase the cost. The height of the upper striped area is $f_1(x) + f_r(0) - \text{size}(f(x)) = f_r(0) - f_r(x)$ for $x \in [0, m)$ and the height of the lower striped area is $\text{size}(f(x)) - f_r(0)$ for $x \in [m, 1)$. All functions are stepwise-constant, but are drawn here using straight downward lines for simplicity; also, the rightmost dark-gray part will "fall down" (forming a $(1-m) \times f_r(0)$ rectangle).

4.4.4 The Main Transformation

In this section, we describe the central transformation in our analysis. It uses liquification and rearranges elements in f and g so as to obtain two properties: that f_r is constant and that $f_1 = g_1$. (The process is explained in Figure 4.2, and a resulting CFP is shown in the upper part of Figure 4.3.) This greatly simplifies the setting and brings us quite close to our ideal CFP (depicted in the lower part of Figure 4.3).

Lemma 4.4.2. *If (f, g) is a CFP with f_1 and f_r non-increasing and $\text{size}(f(1)) \geq f_r(0)$, then there exists another CFP (f', g') with $\frac{cost(f)}{cost(g)} \leq \frac{cost(f')}{cost(g')}$ such that:*

(a) f'_r is constant.

(b) $f'_1 = g'_1$ and it is non-increasing.

(c) There exists $m \in [0, 1]$ such that:

- *for $x \in [0, m)$, $f'(x)$ has liquid elements and exactly one solid element,*
- *for $x \in [m, 1)$, $f'(x)$ has only liquid elements.*

Proof. We begin with an overview of the proof. See also Figure 4.2.

Our procedure to obtain such a CFP has three stages:

1. We liquify all elements except for the largest element of every pattern $f(x)$ for $x \in [0, m)$, for a certain threshold m; at this point, $f_r(x)$ becomes essentially the size of liquid elements in $f(x)$ for $x \in [0, 1)$ (as no pattern contains two solid elements).

2. We move liquid elements in f from right to left, i.e., from $f(y)$ for $y \in [m, 1)$ to $f(x)$ for $x \in [0, m)$ (which only increases the cost of f), so as to make f_r constant.

3. We rearrange elements in g so as to satisfy the condition $f_1' = g_1'$.

Now let us proceed to the details. First, we explain how to choose the threshold m. It is defined so as to ensure that after the liquification, patterns in f have liquid elements of total size $f_r(0)$ on average. Thus we can make all $f_r(x)$ equal to $f_r(0)$. More importantly, we can do this by only moving the liquid elements "higher", so that the cost of f only goes up. (See Figure 4.2 for an illustration of this move.)

More precisely, $m \in [0, 1]$ is chosen so that

$$\int_0^m f_r(0) - f_r(x)\, dx = \int_m^1 \text{size}(f(x)) - f_r(0)\, dx \tag{4.1}$$

(which implies $f_r(0) = \int_0^m f_r(x)\, dx + \int_m^1 \text{size}(f(x))\, dx$: the right-hand expression is the size of elements that will be liquified next). Such an m is guaranteed to exist by non-negativity of the functions under both integrals in (4.1), which follows from our assumptions on f (we have $\text{size}(f(x)) \geq \text{size}(f(1)) \geq f_r(0)$, because $\text{size}_f = f_1 + f_r$ is non-increasing as f_1 and f_r are both non-increasing).

Now we can perform the sequence of steps:

1. The liquification: for each element $p > 0$ which appears in a pattern $f(x) \setminus \{f_1(x)\}$ for $x \in [0, m)$ or in a pattern $f(x)$ for $x \in [m, 1)$, we liquify all its such occurences, as well as their counterparts in g.[9] We have a bound on the ratio of costs by Fact 4.4.2, and f_1 remains non-increasing.

2. Rearranging liquid elements in f: while there exists $x \in [0, m)$ with $f_r(x) < f_r(0)$, find $y \in [m, 1)$ with $f_r(y) > f_r(0)$ and move a liquid element from $f(y)$ to $f(x)$.[10] (As we want to make f_r constant and equal to $f_r(0)$, we move elements from where f_r is too large to where it is too small. Note that as we liquified all elements in $f(x)$ for $x \in [m, 1)$, now size_f is almost equal to f_r on $[m, 1)$.) Once this process is complete, f_r will be constant by the definition of m.[11] (See the right side of Figure 4.2.) Note that size_f was

[9] Formally, let $I_f = \{x \in [0, m) : p \in f(x) \setminus \{f_1(x)\}\} \cup \{x \in [m, 1) : p \in f(x)\}$ and $I_g = \{y \in [0, 1) : p \in g(y)\}$; by compatibility of f and g, the measure of I_g is no less than that of I_f. We liquify p in I_f and in a subset of I_g of the right measure. (If there were patterns where p appeared multiple times, we repeat.)

[10] Formally, let $\tau > 0$ be such that $f_r(x') < f_r(0)$ for $x' \in [x, x + \tau)$ and $x + \tau \leq m$ and $f_r(y') > f_r(0)$ for $y' \in [y, y + \tau)$. Move the liquid elements between these patterns $f(y')$ and $f(x')$.

[11] As we operate in the limit $\varepsilon \to 0$, we ignore issues such as f_r being $\pm\varepsilon$ off.

non-increasing at the beginning of this step, hence we are always moving elements from patterns of smaller total size to patterns of larger total size – This can only increase the cost of f, as we are moving liquid elements from $f(y)$ to $f(x)$ where $x < m \le y$, and therefore $\text{size}(f(x)) \ge \text{size}(f(y))$ (if we consider the liquid element went last in both patters, its contribution to the cost is larger when it moves to the larger pattern). Thus, the ratio of costs increases. This step preserves f_1.

3. Rearranging elements in g: At this point, as f and g are compatible, g only has solid jobs that appear as $f_1(x)$ for $x \in [0, m)$, but they might be arranged in g differently than in f. We want them to have the same arrangement (i.e., $f_1' = g_1'$) so that we can compare f to g more easily. As a first step, we make sure that every pattern in g has at most one solid element. To this end, while there exists $x \in [0,1)$ such that $g(x)$ contains two or more solid elements, let $p > 0$ be one of them, and find $y \in [0,1)$ such that $g(y)$ contains only liquid elements[12]. Now there are two cases: if $\text{size}(g(y)) \ge p$, then we move p to $g(y)$ and move liquid elements of the same total size back to $g(x)$. This preserves $\text{cost}(g)$ (think that these elements went first in the linear orders on both $g(x)$ and $g(y)$). If $\text{size}(g(y)) < p$, then we move p to $g(y)$ and move all liquid elements from $g(y)$ to $g(x)$. This even decreases $\text{cost}(g)$.[13]

   At this point, f and g have the same solid elements, each of which appears as the only solid element in patterns containing it in both f and g. We can now sort g so that for each solid element $p > 0$, it appears in the same positions in f and in g. This operation preserves $\text{cost}(g)$, and thus the entire third step does not decrease the ratio of costs.

□

4.4.5 The Final Form

In the last transformation, we will guarantee that, in g, each large element is the only member of a pattern that contains it. (Intuitively, we do this because such CFPs are the ones which maximize the ratio of costs.) Specifically, we prove Lemma 4.4.3, a strengthened version of Lemma 4.4.2; note that condition *(c')* below is stronger than condition *(c)* of Lemma 4.4.2 (there, g' could have had patterns with both liquid and solid elements), and that condition *(d)* is new. Figure 4.3 shows the difference between CFPs postulated by Lemmas 4.4.2 and 4.4.3.

Lemma 4.4.3. *Let (f, g) be as postulated in Lemma 4.4.2. Then there exists another CFP (f', g') such that:*

(a) f_r' is constant.

[12] Such a y exists because f and g are compatible: the total measure of patterns with solid elements in f is m and these patterns contain only one solid element each, so g must have patterns with no solid elements.

[13] Formally, select $\tau > 0$ so that g is constant on $[x, x+\tau)$ and on $[y, y+\tau)$; for $x' \in [x, x+\tau)$, replace $g(x')$ with $g(x') \setminus \{p\} \cup g(y)$, and for $y' \in [y, y+\tau)$, replace $g(y')$ with $\{p\}$. The cost of g then decreases by $\tau(\text{size}(g(x)) - p)(p - \text{size}(g(y))) > 0$.

(b) $f'_1 = g'_1$.

(c') *There exists* $t \in [0,1]$ *such that:*

- *for* $x \in [0,t)$, $f'(x)$ *has liquid elements and precisely one solid element,*
- *for* $x \in [t,1)$, $f'(x)$ *has only liquid elements,*
- *for* $x \in [0,t)$, $g'(x)$ *has precisely one solid element (and no liquid ones),*
- *for* $x \in [t,1)$, $g'(x)$ *has only liquid elements.*

(d) *The function* $\text{size}_{g'}$ *is constant on* $[t,1)$ *(i.e., the liquid part of* g *is uniform).*

Moreover,

$$\frac{cost(f')}{cost(g')} \geq \min\left(2, \frac{cost(f)}{cost(g)}\right).$$

Proof. Let us begin by giving a proof outline; see also Figure 4.3 for an illustration.

- Our primary objective is to make g_1 equal to size_g on $[0,t)$ (where t is to be determined). To this end, we increase the sizes of the solid elements in $g(x)$ for $x \in [0,t)$ and while we decrease the total sizes of liquid elements for these $g(x)$ (which keeps size_g unchanged). To offset this change, we decrease the sizes of solid elements in $g(y)$ for $y \in [t,m)$ (and also increase the total sizes of liquid elements there). We also modify the sizes of solid elements in f so as to keep f and g compatible and preserve the properties *(a)*-*(b)*.
- The threshold t is defined so that after we finish this process, the solid elements in $g(x)$ for $x \in [0,t)$ will have filled out the entire patterns $g(x)$, and the solid elements in $g(y)$ for $y \in [t,m)$ will have disappeared.
- Our main technical claim is that this process does not invalidate the ratio of costs.
- Finally, we can easily ensure condition *(d)* by levelling g on $[t,1)$, which only decreases its cost.

Now we proceed to the details. First we define the threshold $t \in [0,m]$ as a solution to the equation

$$\int_0^t g_r(x)\,dx = \int_t^m g_1(x)\,dx,$$

which exists because the functions under both integrals are nonnegative. The left-hand side will be the total increase in sizes of solid elements in $g(x)$ for $x \in [0,t)$ and the right-hand side will be the total decrease in sizes of solid elements in $g(y)$ for $y \in [t,m)$ (these elements will disappear completely).

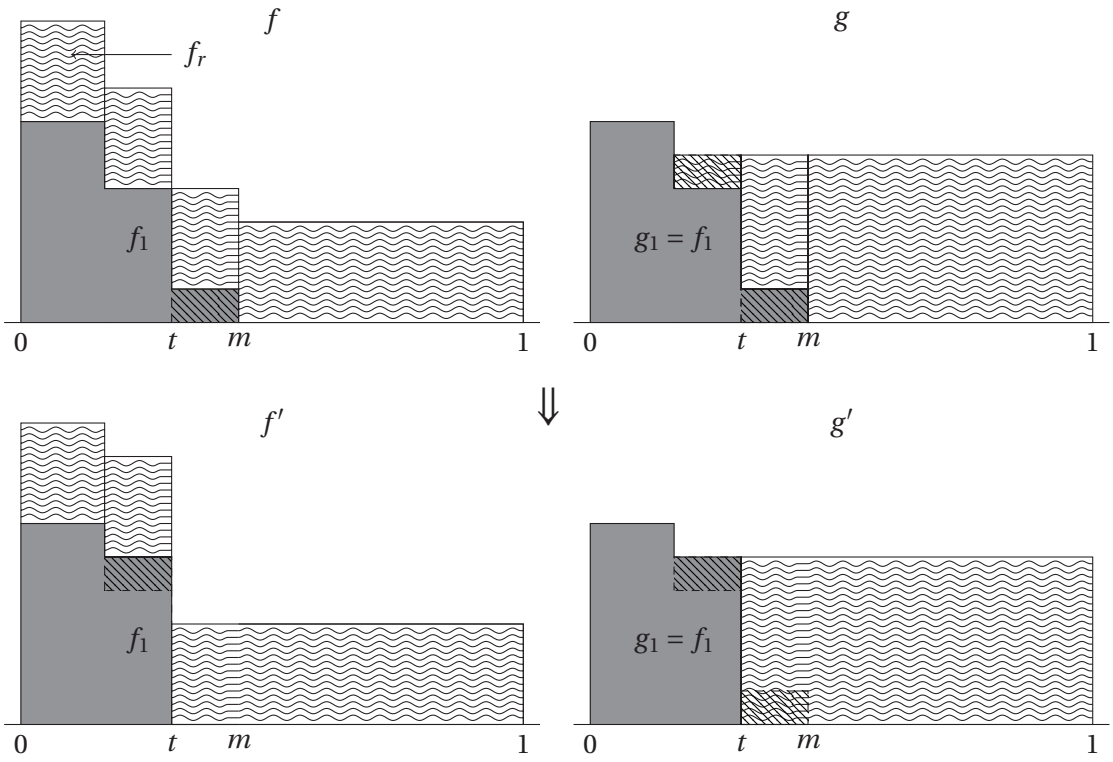

Figure 4.3: An example of two CFPs: (f, g) is produced by Lemma 4.4.2, whereas (f', g') is produced by Lemma 4.4.3. In this picture, the height of the plot corresponds to size$(f(x))$ for each x, and the shaded and wavy parts correspond to the contributions of f_1 and f_r to size$_f$; similarly for g. The wavy parts are liquid. In Lemma 4.4.3 we want to make $f_1 = g_1$ equal to size$_g$ on an interval $[0, t)$, so we increase sizes of solid elements in g on that interval, while we decrease those on the interval $[t, m)$. The striped regions in the pictures of g correspond to these changes. (We repeat the same changes in f, and we also move liquid elements in g to keep size$_g$ unchanged.) The threshold $t \in [0, m]$ is chosen so that g_1 becomes equal to size$_g$ on $[0, t)$ and the solid elements on $[t, m)$ are eradicated (i.e., so that the areas of the striped regions in g are equal).

Here we will carry out the process that we announced in the outline. Specifically, while there exists $x \in [0, t)$ with $g_r(x) > 0$, do the following:

- find $y \in [t, m)$ with $g_1(y) > \varepsilon$ (i.e., $g(y)$ where the solid element has not been eradicated yet),
- increase the size of the solid element in $g(x)$ by ε,
- do the same in f,
- decrease the size of the solid element in $g(y)$ by ε,
- do the same in f,

- move one liquid element (of size ε) from $g(x)$ to $g(y)$.

Formally, as usual, we find $\tau > 0$ such that f and g are constant on $[x, x+\tau)$ and on $[y, y+\tau)$ and we do this in all of these patterns.

Note that the following invariants are satisfied after each iteration:

- $f_1 = g_1$,
- f_r does not change,
- size_g does not change,
- $\int_0^m g_1(x)\,dx$ does not change,
- g_1 can only increase on $[0, t)$ and it can only decrease on $[t, m)$,
- $f_1(x) \geq f_1(y)$ for all $x \in [0, t)$ and $y \in [t, m)$.[14]

By the definition of t, when this process ends, the patterns $g(x)$ for $x \in [0, t)$ contain only a single solid element, whereas $g(x)$ for $x \in [t, m)$ (thus also for $x \in [t, 1)$) contain no solid elements. Since $f_1 = g_1$, the patterns $f(x)$ also have only liquid elements for $x \in [t, 1)$. Thus properties *(a), (b)* and *(c')* are satisfied. We reason about the ratio of costs in the following two technical claims:

Claim 4.4.1. *In a single iteration, cost*(f) *increases by* 2α *and cost*(g) *increases by* α*, for some* $\alpha \geq 0$.

Proof. The patterns have changed so that:

- $f(x)$ had $f_1(x)$ increased by ε,
- $f(y)$ had $f_1(y)$ decreased by ε,
- $g(x)$ had $g_1(x)$ increased by ε and one liquid element removed,
- $g(y)$ had $g_1(y)$ decreased by ε and one liquid element added.

Since the order of elements does not matter, in computing $\text{cost}(f)$ we think that the solid element goes last in the linear order:

$$\Delta\,\text{cost}(f(x)) = (f_1(x)+\varepsilon)(\text{size}(f(x))+\varepsilon) - f_1(x)\,\text{size}(f(x)) = \varepsilon\left(\text{size}(f(x)) + f_1(x) + \varepsilon\right),$$
$$\Delta\,\text{cost}(f(y)) = (f_1(y)-\varepsilon)(\text{size}(f(y))-\varepsilon) - f_1(y)\,\text{size}(f(y)) = \varepsilon\left(-\text{size}(f(y)) - f_1(y) + \varepsilon\right),$$

[14] This is because $f_1 = g_1$ was initially non-increasing and since then it has increased on $[0, t)$ and decreased on $[t, m)$.

$$\begin{aligned}\Delta\text{cost}(f(x)) + \Delta\text{cost}(f(y)) &= \varepsilon\left(\text{size}(f(x)) - \text{size}(f(y)) + f_1(x) - f_1(y) + 2\varepsilon\right)\\ &= 2\varepsilon\left(f_1(x) - f_1(y) + \varepsilon\right),\end{aligned}$$

where in the last line we used that $\text{size}(f(x)) - \text{size}(f(y)) = f_1(x) + f_r(x) - f_1(y) - f_r(y) = f_1(x) - f_1(y)$ since f_r is constant.

In computing $\text{cost}(g)$, we think that the solid element and the one liquid element that was added or removed go first (and other elements are unaffected since size_g is preserved):

$$\begin{aligned}\Delta\text{cost}(g(x)) &= (g_1(x) + \varepsilon)^2 - (g_1(x)^2 + \varepsilon(g_1(x) + \varepsilon)) = \varepsilon g_1(x),\\ \Delta\text{cost}(g(y)) &= ((g_1(y) - \varepsilon)^2 + \varepsilon g_1(y)) - g_1(y)^2.\end{aligned}$$

Adding up, we have:

$$\Delta\text{cost}(g(x)) + \Delta\text{cost}(g(y)) = \varepsilon\left(g_1(x) - g_1(y) + \varepsilon\right) = \varepsilon\left(f_1(x) - f_1(y) + \varepsilon\right),$$

where we used that $f_1 = g_1$. Thus we have that

$$\Delta\text{cost}(f(x)) + \Delta\text{cost}(f(y)) = 2\left[\Delta\text{cost}(g(x)) + \Delta\text{cost}(g(y))\right]$$

and we prove the statement by setting

$$\alpha = \tau\left(\Delta\text{cost}(g(x)) + \Delta\text{cost}(g(y))\right) = \tau\varepsilon\left(f_1(x) - f_1(y) + \varepsilon\right) \geq 0$$

(recall that τ is the fraction of patterns where we increase g_1; non-negativity follows by the last invariant above). □

Claim 4.4.2. *Let (f', g') be the CFP obtained at this point and (f, g) be the original CFP. Then*

$$\frac{cost(f')}{cost(g')} \geq \min\left(2, \frac{cost(f)}{cost(g)}\right).$$

Proof. By Claim 4.4.1, we have $\text{cost}(f') = \text{cost}(f) + 2\beta$ and $\text{cost}(g') = \text{cost}(g) + \beta$ for some $\beta \geq 0$ (which is the sum of α's from Claim 4.4.1). Now there are two cases:

- if $\frac{\text{cost}(f)}{\text{cost}(g)} \leq 2$, then $\frac{\text{cost}(f)+2\beta}{\text{cost}(g)+\beta} \geq \frac{\text{cost}(f)}{\text{cost}(g)}$,
- if $\frac{\text{cost}(f)}{\text{cost}(g)} \geq 2$, then $\frac{\text{cost}(f)+2\beta}{\text{cost}(g)+\beta} \geq 2$ (even though the ratio decreases, it stays above 2).

□

Finally, as the last step, we equalize the total sizes of liquid elements in $g(x)$ for $x \in [t, 1)$ (by moving liquid elements from larger patterns to smaller patterns, until all are equal), thus satisfying property *(d)*. Clearly, this can only decrease the cost of g (by minimizing the variance of size_g on the interval $[t, 1)$), so the ratio increases and Lemma 4.4.3 follows. □

Note that Lemma 4.4.3 does not guarantee that the ratio of costs increases; we only claim that it either increases, or it is now more than 2. However, we will show shortly, in Lemma 4.4.4, that the ratio is actually much below 2, so the latter is in fact impossible.

Now that we have our ideal CFP, we can finally bound its cost ratio.

Lemma 4.4.4. *Given a CFP (f, g) as postulated in Lemma 4.4.3 (see the lower part of Figure 4.3), we have*

$$\frac{cost(f)}{cost(g)} \leq \frac{1+\sqrt{2}}{2}.$$

The proof proceeds in two simple steps: First, we argue that we can assume without loss of generality that there is only a single large element (i.e., $f_1 = g_1$ is constant on $[0, t)$). Then, for such pairs of functions, the ratio is simply a real function of three variables whose maximum is easy to compute.

Proof. For a first step, we assume without loss of generality that there is only a single large element (i.e., $f_1 = g_1$ is constant on $[0, t)$). This is due to the fact that both f and g can be written as a weighted average of functions with a single large element. Formally, let $\ell_1, \ell_2, \ldots$ be the step lengths of f_1 on $[0, t)$, so that $\sum_i \ell_i = t$ and f_1 is constant on $[0, \ell_1)$, on $[\ell_1, \ell_1 + \ell_2)$ and so on. Define f^i to be f with the whole f_1 on $[0, t)$ replaced by the i-th step of f_1, i.e.,

$$f^i(x) = \begin{cases} f(\ell_1 + \ldots + \ell_{i-1}) & \text{for } x \in [0, t), \\ f(x) & \text{for } x \in [t, 1). \end{cases}$$

Define g^i similarly. By inspecting f, we observe that $f_r(x)$ is constant in $[0, 1)$, and equal to $\text{size}(f(x))$ in $[t, 1)$. Hence, $\text{cost}(f(x))$ is constant in $[t, 1)$, whereas in $[0, t)$ it is uniquely determined by $f_1(x)$ and $\text{size}(f(1))$. Therefore, $\text{cost}(f)$ can be written as

$$\begin{aligned} \text{cost}(f) &= \sum_i \ell_i \,\text{cost}(f(\ell_1 + \ldots + \ell_{i-1})) + (1-t) \cdot \text{cost}(f(1)) \\ &= \sum_i \frac{\ell_i}{t} \left[t \cdot \text{cost}(f(\ell_1 + \ldots + \ell_{i-1})) + (1-t) \cdot \text{cost}(f(1)) \right] \\ &= \sum_i \frac{\ell_i}{t} \,\text{cost}(f^i) \end{aligned}$$

and similarly $\text{cost}(g) = \sum_i \frac{\ell_i}{t} \,\text{cost}(g^i)$. Thus, if we have $\frac{\text{cost}(f^i)}{\text{cost}(g^i)} \leq \frac{1+\sqrt{2}}{2}$ for each i, then

$$\frac{\text{cost}(f)}{\text{cost}(g)} = \frac{\sum_i \frac{\ell_i}{t} \,\text{cost}(f^i)}{\sum_i \frac{\ell_i}{t} \,\text{cost}(g^i)} \leq \frac{\sum_i \frac{\ell_i}{t} \frac{1+\sqrt{2}}{2} \,\text{cost}(g^i)}{\sum_i \frac{\ell_i}{t} \,\text{cost}(g^i)} = \frac{1+\sqrt{2}}{2}.$$

So we assume that f_1 is constant on $[0, t)$ (i.e., the shaded areas in Figure 4.3 are rectangles). Let $\gamma = f_1(0)$ be the large element and λ be the total mass of liquid elements (the same in f as

in g), i.e., $\lambda = f(1) = (1-t)g(1)$. In the limit $\varepsilon \to 0$ we have

$$\frac{\text{cost}(f)}{\text{cost}(g)} = \frac{t\left(\gamma^2 + \int_0^\lambda (\gamma + x)\,dx\right) + (1-t)\int_0^\lambda x dx}{t\gamma^2 + (1-t)\int_0^{g(1)} x\,dx} = \frac{t\left(\gamma^2 + \gamma\lambda + \frac{\lambda^2}{2}\right) + (1-t)\frac{\lambda^2}{2}}{t\gamma^2 + (1-t)\frac{\left(\frac{\lambda}{1-t}\right)^2}{2}} = \frac{t\gamma^2 + t\gamma\lambda + \frac{\lambda^2}{2}}{t\gamma^2 + \frac{\lambda^2}{2(1-t)}}$$

and we need to prove that this expression is at most $\frac{1+\sqrt{2}}{2}$ for all $t \in [0,1)$, $\gamma \geq 0$ and $\lambda \geq 0$. So we want to show

$$t\gamma^2 + t\gamma\lambda + \frac{\lambda^2}{2} \leq \frac{1+\sqrt{2}}{2}\left(t\gamma^2 + \frac{\lambda^2}{2(1-t)}\right),$$

that is,

$$\lambda^2\left(\frac{1+\sqrt{2}}{4(1-t)} - \frac{1}{2}\right) - \lambda \cdot t\gamma + \frac{\sqrt{2}-1}{2}t\gamma^2 \geq 0.$$

Note that $\frac{1+\sqrt{2}}{4(1-t)} - \frac{1}{2} > 0$ for $t \in [0,1)$, so this is a quadratic polynomial in λ whose minimum value (over $\lambda \in \mathbb{R}$) is

$$\frac{\sqrt{2}-1}{2}t\gamma^2 - \frac{t^2\gamma^2}{4\left(\frac{1+\sqrt{2}}{4(1-t)} - \frac{1}{2}\right)} = t\gamma^2\left(\frac{\sqrt{2}-1}{2} - \frac{t}{\frac{1+\sqrt{2}}{1-t} - 2}\right)$$

and we should prove that this is non-negative. If $t = 0$ or $\gamma = 0$, then this is clearly true; otherwise we multiply both sides of the inequality by $\frac{1-t}{t\gamma^2}\left(\frac{1+\sqrt{2}}{1-t} - 2\right)$ (a positive number) and after some calculations we are left with showing

$$t^2 + \left(\sqrt{2}-2\right)t + \frac{3-2\sqrt{2}}{2} \geq 0$$

but this is again a quadratic polynomial, whose minimum is 0. □

To conclude the proof of Theorem 4.1.1, we require the following lemma regarding the tightness of our analysis of Algorithm 4.1, which we will prove shortly afterwards:

Lemma 4.4.5. *For any $\delta > 0$, there is an instance I of $R||\sum p_j C_j$ whose optimal value is c, and an optimal configuration-LP solution whose objective value is also c, such that the rounded solution returned by Algorithm 4.1 has cost at least $(\frac{1+\sqrt{2}}{2} - \delta)c$.*

We conclude the proof of Theorem 4.4.1:

Proof of Theorem 4.4.1. It is straightforward to see that

$$\frac{1+\sqrt{2}}{2} \geq \frac{\text{cost}(f')}{\text{cost}(g')} \geq \min\left(2, \frac{\text{cost}(f)}{\text{cost}(g)}\right) = \min\left(2, \frac{\sum_i y_i^{\text{out}} \text{cost}(C_i)}{\sum_i y_i^{\text{in}} \text{cost}(C_i)}\right)$$

where (f,g) is produced from $(y^{\text{out}}, y^{\text{in}})$ as in Fact 4.4.1 and (f',g') is produced from (f,g) by applying Lemmas 4.4.1, 4.4.2 and 4.4.3; the first inequality is by Lemma 4.4.4. It follows that either $\frac{1+\sqrt{2}}{2} \geq 2$ (false) or $\frac{1+\sqrt{2}}{2} \geq \frac{\sum_i y_i^{\text{out}} \text{cost}(C_i)}{\sum_i y_i^{\text{in}} \text{cost}(C_i)}$. Together with Lemma 4.4.5, Theorem 4.4.1 follows. □

Let us now prove Lemma 4.4.5:

Proof of Lemma 4.4.5. The intuitive explanation is that the bound in Lemma 4.4.4 is tight, hence there exists a CFP in the final form (as postulated by Lemma 4.4.3) with ratio exactly $\frac{1+\sqrt{2}}{2}$. Furthermore, this CFP indeed almost corresponds to an instance of $R||\sum p_j C_j$ (except for the fact that the parameters that maximize the ratio are irrational). We make this intuition formal in the following.

Let

$$h(t,\gamma,\lambda) = \frac{t\gamma^2 + t\gamma\lambda + \frac{\lambda^2}{2}}{t\gamma^2 + \frac{\lambda^2}{2(1-t)}}$$

be the function specifying the ratio of CFPs in the final form. In Lemma 4.4.4, we proved that $h(t,\gamma,\lambda) \leq \frac{1}{2} + \frac{1}{\sqrt{2}}$ for all $t,\gamma,\lambda \in [0,1]$. To begin, let us fix the following maximizer $(t^\star,\lambda^\star,\gamma^\star)$ of h: $t^\star = 1 - \frac{1}{\sqrt{2}}$, $\gamma^\star = \frac{1}{2}$ and $\lambda^\star = \frac{\sqrt{2}-1}{2}$; we have $h(t^\star,\lambda^\star,\gamma^\star) = \frac{1}{\sqrt{2}} + \frac{1}{2}$.

Let us choose a small rational η. Next, let us fix rationals $\tilde{t} \in [t^\star - \eta, t^\star]$, $\tilde{\lambda} \in [\lambda^\star - \eta, \lambda^\star]$ and $\tilde{\gamma} \in [\gamma^\star, \gamma^\star + \eta]$ such that $h(\tilde{t},\tilde{\gamma},\tilde{\lambda}) \geq \frac{1+\sqrt{2}}{2} - \eta$. Then, there exist positive integers k, T and Λ such that $T = \tilde{t}k$ and $\Lambda = k\tilde{\lambda}$. Finally, select a small rational $\epsilon \leq \eta$ such that $\varepsilon = \frac{\tilde{\lambda}}{k_1(1-\tilde{t})}$, for some integer $k_1 > 0$, and $\varepsilon = \frac{\tilde{\lambda}}{k_2}$, for some integer $k_2 > 0$.

Next, we create an instance I_ε of $R||\sum p_j C_j$ which consists of k machines, a set $\mathcal{T}$ of T jobs of size $\tilde{\gamma}$ each, and a set $\mathcal{L}$ of Λ/ε jobs of size ε each; any job can be assigned to any machine. An optimal solution to this instance will assign the jobs from $\mathcal{T}$ alone on T machines, and distribute the jobs from $\mathcal{L}$ evenly on the rest of the machines (i.e., these machines will all receive $\frac{\tilde{\lambda}}{(1-\tilde{t})\varepsilon}$ jobs of size ε each). The fact that this is an optimal solution follows in a straightforward manner from the following two observations:

- A solution that assigns a job from $\mathcal{L}$ on the same machine as a job from $\mathcal{T}$ is sub-optimal: indeed, the average makespan is less than $\tilde{\gamma}$ (in fact, it is exactly $\tilde{t}\tilde{\gamma} + \tilde{\lambda}$, which is at most $\tilde{\gamma}$, due to the fact that $t^\star\gamma^\star + \lambda^\star < \gamma^\star$ and due to the intervals which we choose $\tilde{t}$, $\tilde{\gamma}$ and $\tilde{\lambda}$ from), which implies that we can always reassign such a job to a machine with smaller makespan, thus decreasing the solution cost.

- Similarly, in any optimal solution, jobs from $\mathcal{T}$ are not assigned on the same machine.

Now, consider the configuration-LP solution y_ε that assigns to every machine a configuration that consists of a single job from $\mathscr{T}$ (i.e., of cost $\tilde{\gamma}^2$) with probability $\tilde{t}$, and a configuration that consists of $\frac{\tilde{\lambda}}{(1-\tilde{t})\varepsilon}$ jobs from $\mathscr{L}$ (i.e., of cost $\sum_{1\le i\le \frac{\tilde{\lambda}}{(1-\tilde{t})\varepsilon}} \sum_{1\le j<i} \varepsilon^2 = \frac{\tilde{\lambda}^2}{2(1-\tilde{t})^2} + \frac{\tilde{\lambda}}{2(1-\tilde{t})}\varepsilon$) with probability $1-\tilde{t}$. Clearly, the cost of this LP solution is equal to that of any optimal integral solution (in fact, the LP solution is a convex combination of all integral optimal solutions). Furthermore, this LP solution is optimal (we can see this by applying the reasoning we used for the integral optimum to all the configurations in the support of a fractional solution).

Algorithm 4.1 will assign to any machine a configuration that consists of a single job from $\mathscr{T}$ and $\tilde{\lambda}/\varepsilon$ jobs from $\mathscr{L}$ (i.e., of cost $\tilde{\gamma}(\tilde{\gamma}+\tilde{\lambda}) + \sum_{1\le i\le \frac{\tilde{\lambda}}{\varepsilon}} \sum_{1\le j<i} \varepsilon^2 = \tilde{\gamma}(\tilde{\gamma}+\tilde{\lambda}) + \frac{\tilde{\lambda}^2}{2} + \frac{\tilde{\lambda}}{2}\varepsilon$) with probability $\tilde{t}$, and a configuration that consists of $\tilde{\lambda}/\varepsilon$ jobs from $\mathscr{L}$ (i.e., of cost $\sum_{1\le i\le \frac{\tilde{\lambda}}{\varepsilon}} \sum_{1\le j<i} \varepsilon^2 = \frac{\tilde{\lambda}^2}{2} + \frac{\tilde{\lambda}}{2}\varepsilon$) with probability $1-\tilde{t}$. To see this, first observe that every machine has a total fractional assignment of jobs from $\mathscr{T}$ equal to $\tilde{t}$, and a total fractional assignment of jobs from $\mathscr{L}$ equal to $\frac{\tilde{\lambda}}{\varepsilon}$. Therefore, the first bucket created by Algorithm 4.1 for any machine will contain a $\tilde{t}$-fraction of $\mathscr{T}$-jobs and an $(1-\tilde{t})$-fraction of $\mathscr{L}$-jobs, and the rest of the buckets will be filled up with $\mathscr{L}$-jobs (since $\frac{\tilde{\lambda}}{\varepsilon}$ is an integer, the last bucket will be filled up to a $\tilde{t}$-fraction). This implies that, in a worst-case output distribution, with probability $\tilde{t}$ any machine receives a $\mathscr{T}$-job and $\mathscr{L}$-jobs of total size $\tilde{\lambda}$, and with probability $(1-\tilde{t})$ it receives $\mathscr{L}$-jobs of total size $\tilde{\lambda}$.

Now, the ratio of the expected cost of the returned solution to the LP cost, for any machine, is then

$$\frac{\tilde{t}\tilde{\gamma}^2 + \tilde{t}\tilde{\gamma}\tilde{\lambda} + \frac{\tilde{\lambda}^2}{2} + \frac{\tilde{\lambda}}{2}\varepsilon}{\tilde{t}\tilde{\gamma}^2 + \frac{\tilde{\lambda}^2}{2(1-\tilde{t})} + \frac{\tilde{\lambda}}{2}\varepsilon} \ge \frac{\tilde{t}\tilde{\gamma}^2 + \tilde{t}\tilde{\gamma}\tilde{\lambda} + \frac{\tilde{\lambda}^2}{2}}{\tilde{t}\tilde{\gamma}^2 + \frac{\tilde{\lambda}^2}{2(1-\tilde{t})} + \tilde{\lambda}\frac{\varepsilon}{2}} = h(\tilde{t},\tilde{\gamma},\tilde{\lambda}) \frac{\tilde{t}\tilde{\gamma}^2 + \frac{\tilde{\lambda}^2}{2(1-\tilde{t})}}{\tilde{t}\tilde{\gamma}^2 + \frac{\tilde{\lambda}^2}{2(1-\tilde{t})} + \frac{\tilde{\lambda}}{2}\varepsilon}$$

which is at least $\frac{1+\sqrt{2}}{2} - \delta$ if we pick ε and η small enough; since the cost of the LP solution is equal to that of any optimal integral solution, the claim follows. □

It is interesting to note that, given the instance and LP solution from the proof of Lemma 4.4.5, any random assignment produced by Algorithm 4.1 will assign the same amount of small jobs to all the machines, whereas it will assign a large job to a $\tilde{t}$-fraction of the machines. Therefore derandomizing Algorithm 4.1 by picking the *best possible* matching (instead of picking one at random) will not improve upon the $\frac{1+\sqrt{2}}{2}$ ratio.

Theorem 4.4.1 and Lemma 4.4.5 together imply Theorem 4.1.1.

4.5 Integrality Gap Lower Bound

First of all, observe that Theorem 4.1.1, apart from establishing the existence of a 1.21-approximation algorithm for $R||\sum_j p_j C_j$, also implies an upper bound on the integrality gap of its configuration-LP. Hence, we accompany our main result with a lower bound on the integrality gap of the configuration-LP for $R||\sum_j p_j C_j$:

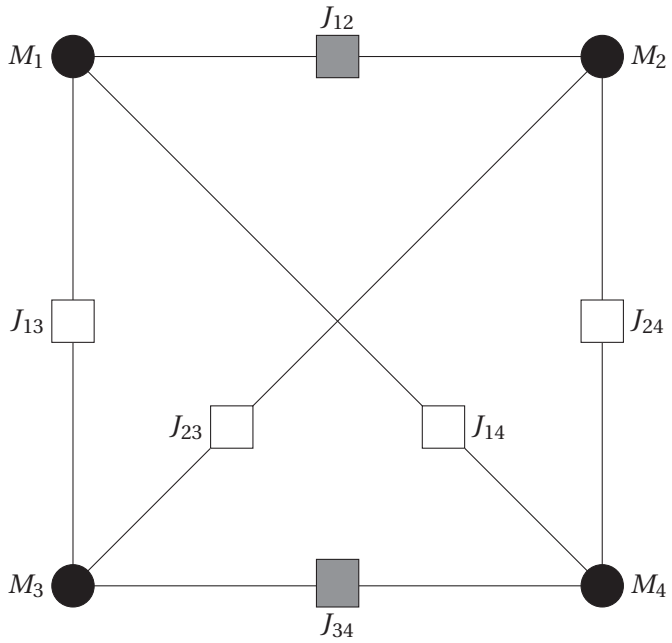

Figure 4.4: The $\frac{13}{12}$-integrality gap instance. In this picture, black circles correspond to machines, gray boxes correspond to jobs of size 3, and white boxes correspond to jobs of size 1. An edge between a circle and a box means that the corresponding job can be assigned to the corresponding machine.

Theorem 4.1.3. *The integrality gap of the configuration-LP for* $R||\sum p_j C_j$ *is at least* 13/12.

Proof. Consider the following instance on 4 machines M_1, M_2, M_3, M_4: for every pair $\{M_i, M_j\}$ of machines there is one job J_{ij} that can be processed only on these two machines. Jobs J_{12} and J_{34} are *large*: they have weight and size 3, whereas the other four jobs are *small* and have weight and size 1. (See Figure 4.4 for an illustration.)

First we show that any integral schedule has cost at least 26. Without loss of generality, the large job J_{12} is assigned to machine M_1 and the other large job J_{34} is assigned to machine M_3. The small job J_{13} must also be assigned to one of them, say to M_1. This costs $9+9+4=22$. The remaining three small jobs J_{12}, J_{14} and J_{24} cannot all be assigned to distinct machines with zero makespan (since only M_2 and M_4 are such), so they will incur at least $1+1+2=4$ units of cost.

The configuration-LP has a solution of cost 24. Specifically, it can assign to each machine M_i two configurations, each with fractional value $\frac{1}{2}$: the singleton large job that can be processed on that machine, or the two small jobs that can be processed on that machine. Then each job is processed with fractional value $\frac{1}{2}$ by each of the two machines that can process it. The cost is $4 \cdot (\frac{1}{2} \cdot 9 + \frac{1}{2} \cdot (1+2)) = 24$. Thus the integrality gap is at least $\frac{26}{24} = \frac{13}{12} > 1.08$. □

4.6 Bi-objective Approximation Algorithm

To begin with, in order to design a bi-objective approximation algorithm, we will need a slightly different configuration-LP than the one we used for the min-sum of weighted completion times problem. In particular, we need to restrict the support of our fractional solution to only include configurations that respect the makespan constraint. Let $\mathscr{C}_i = \{C \subseteq \mathscr{J}_i : \sum_{j \in C} p_j \leq T\}$; we have the following configuration-LP relaxation:

$$\begin{aligned}
\min \quad & \sum_{i \in \mathscr{M}} \sum_{C \in \mathscr{C}_i} y_{iC} \operatorname{cost}(C) \\
\text{s.t.} \quad & \sum_{C \in \mathscr{C}_i} y_{iC} \leq 1 && \forall i \in \mathscr{M}, \\
& \sum_{i \in \mathscr{M}} \sum_{C \in \mathscr{C}_i : j \in C} y_{iC} = 1 && \forall j \in \mathscr{J}, \\
& y_{iC} \geq 0 && \forall i \in \mathscr{M},\ C \in \mathscr{C}_i .
\end{aligned}$$

Again, we do not know how to solve this LP exactly, but we can approximately solve it up to any desired degree of accuracy (see Appendix A).

The algorithm behind Theorem 4.1.2 is simply the application of Algorithm 4.1 to an (approximately) optimal fractional solution of the modified configuration-LP. Then, Theorem 4.1.1 implies the cost guarantee of Theorem 4.1.2, whereas the original analysis of Shmoys and Tardos [39] implies the makespan guarantee.

5 Conclusions and Future Directions

In this final chapter, we discuss the main conclusions of this thesis and the possible directions in which the results and techniques we provided can be extended.

5.1 Restricted Max-Min Fair Allocation

For the max-min fair allocation problem, we designed a polynomial-time $\frac{1}{13}$-approximation algorithm. We did this by designing a local-search procedure that iteratively extended a partial solution to include one more agent. The essential part of the design of this procedure is that, throughout its execution, we made sure to update our partial solution only when this update would affect a significant fraction of the involved agents. This design feature enabled us to place a polynomial upper bound on the number of different states the local search might enter.

5.1.1 Future Directions

When we study an **NP**-hard optimization problem, our main goal is to achieve matching approximability and inapproximability results. Although the restricted max-min fair allocation problem is **NP**-hard to approximate within a factor better than $1/2$, in this thesis we showed the problem can be approximated within a factor of $1/13$ in polynomial time. Towards achieving tight results, our work, and the techniques we used, point out two main directions: (1) improving the approximation guarantee that we can achieve in polynomial time by using the configuration-LP, and (2) improving the best known lower bound on its integrality gap.

Polynomial Time Algorithms In this direction, we need first to identify, in our approach, the bottleneck that does not enable our ideas to produce a better approximation guarantee. After inspecting our algorithm and its analysis, we can observe that the main bottleneck appears in our refutation of the configuration-LP. Specifically, when we design a solution to the dual LP that certifies the primal is infeasible, we have to "buy" all the items that appeared in the local

search, whereas the only agents that "pay" for them are those for which our local search used almost all of the items they can receive.

Previous approaches [32, 2] refuted the configuration-LP by having all agents that appear in the local search pay for the items. This is the reason these approaches produce our best-known bounds on the integrality gap of the configuration-LP. Our approach, however, uses only a fraction of the agents to pay for the bought items (on the technical level, this is a result of our design, because we introduce one addable edge per blocking edge, instead of many, and we only update our matching when a layer has many immediately addable edges, instead of one). This fact ultimately implies that we have to include fewer items per agent in our local search, which implies that the value of the items each agent receives is reduced. Therefore, the main question that arises is, How can we modify our approach in a way that ensures almost all agents that are included in our local search pay for the items we use? In order to achieve this, we might have to ensure that every agent receives multiple addable edges, while maintaining a meaningful condition of updating the partial solution, i.e., a condition that will imply every update implies significant progress.

Integrality Gap The second question that arises is, How can one achieve an improved bound on the integrality gap of the configuration-LP for the restricted max-min fair allocation problem? In this case, it is less clear what the most promising direction is. At this moment, our best bounds are provided by local-search algorithms that run in super-polynomial time and use the configuration-LP to argue that failure implies the guessed value is infeasible. Although it is not clear how we can modify these approaches to achieve improved bounds (we would have to ensure we include more items in our local search, and that the unassigned items in the neighbourhood of each agent are less), one interesting and unexplored direction is to directly round the configuration-LP. Although there is little understanding of how this task can be carried out, successfully taking this approach might open up many interesting directions for a multitude of allocation problems.

5.2 Min-Sum of Weighted Completion Times

For the min-sum of weighted completion times problem with uniform Smith ratios, we provided a polynomial-time rounding algorithm for the configuration-LP that achieves a $\frac{1+\sqrt{2}}{2}$-approximation guarantee. We did this is by using a well-known rounding algorithm for allocation problems, which achieves a strong concentration on the number of jobs assigned to each machine. Furthermore, this rounding algorithm also implicitly introduces negative correlations on the assignment of jobs of similar sizes to each machine. This enabled us to establish the $\frac{1+\sqrt{2}}{2}$-approximation guarantee, through a series of modifications of the output distribution of each machine, where the cost of each intermediate distribution is an upper bound on the cost of the original one.

5.2.1 Future Directions

When examining how we can extend our ideas to the general min-sum of weighted completion times problem, or how we can develop new ones, we have to first identify our available tools and the existing limitations. The configuration-LP is certainly such a strong tool: The fact that it does not underestimate the cost of any assignment is essential, and leaves us only with the problem of extending the local distributions over assignments defined by the configuration-LP to a global distribution.

However, how we can achieve this is far from clear. It is hard to imagine how we can significantly beat the 3/2-ratio by using independent rounding as our basic rounding scheme. Furthermore, if we only modify the order in which the jobs are introduced into the buckets, we will not be able to extend the rounding algorithm that we used to non-uniform Smith ratios. An approach that could conceivably work is to design an algorithm that carefully constructs each bucket separately (e.g., by including in each bucket jobs that increase the cost significantly when assigned on the same machine), rather than by using a simple total order on the jobs.

Even then, it is not completely clear how far such an approach might go, as the rounding algorithm we used inherently introduces very strong negative correlations for very small groups of jobs, whereas it might be required that we do so for larger groups. One promising technique in this direction is also the use of SDP hierarchies (e.g., see [5]); rounding solutions to such relaxations in a way that we preserve some of the correlations the fractional solution implies is a promising idea and is one that might be applicable in many other allocation problems as well.

5.3 General Future Directions

Let us now discuss some research directions that are laid out by our work, but are less relevant to the specific problems we studied. One first such direction is trying to use the configuration-LP in order to produce solutions to other, weaker, but more well-understood LP-relaxations that exhibit some extra useful properties. Then, we can use some already known algorithm to round the solution to the weaker relaxation and use the properties of the configuration-LP to establish an increase in performance. This approach was used to prove that the integrality gap of the configuration-LP for the MBA problem is strictly better than 3/4 [26, 25], which is the integrality gap achieved by the weaker assignment-LP. It is quite possible that this approach can be extended to other similar problems: for example, we can prove that for a special case of the GAP problem, configuration-LP solutions whose support contains configurations with at most two items can be projected to assignment-LP solutions in such a way that a simple modification of the algorithm by Shmoys and Tardos [39] achieves a 2/3-approximation guarantee. Whether or not we can generalize this claim to any configuration-LP solution, and subsequently to more allocation problems, is an intriguing question.

Another direction we would like to point out, is the design of algorithms that directly round

solutions to the configuration-LP. So far, the configuration-LP has mostly been used either in conjuction with some local search algorithm, or in order to design a solution to another relaxation that exhibits some nice properties; one notable exception is the bin packing problem [33, 34]. Although we still lack the understanding of how such a feat is possible, achieving it would, at the very least, contribute to our understanding of the power and limitations of the configuration-LP, and would hopefully provide better approximation results for various allocation problems.

Finally, one general question that deserves some attention is how we can use local search in order to design good approximation algorithms for allocation problems. So far, local search has been instrumental in providing integrality gaps and approximation algorithms for the min-max scheduling problem [44, 24] and the max-min fair allocation problem [2, 32, 1]. On one hand, the hardness of the constraints of these problems make local search a more appealing technique. On the other hand, there is not much known on whether we can apply such techniques on problems with softer constraints and different objective functions (e.g. GAP, MBA). Our understanding of how local search can provide better guarantees than other, more standard, techniques for these problems is limited, but this is certainly an interesting question, whose importance is only amplified by the practical relevance of combinatorial local search algorithms.

A Solving a Configuration-LP

Over the course of this thesis, we established the existence of good approximation algorithms for two different problems. Their existence relied on the existence of a corresponding configuration-LP. On the one hand, our algorithm for the restricted max-min fair allocation problem does not explicitly solve the appropriate configuration-LP, but rather uses it as a lower bound in the accompanying analysis. On the other hand, the configuration-LP is indeed explicitly solved as part of our approximation algorithm for the min-sum of weighted completion times problem. For the sake of completeness, we will be describing formally how to solve the configuration-LP for the restricted max-min fair allocation problem, and for the min-sum of weighted completion times with makespan constraints problem (we do not explain how to solve the configuration-LP for the the min-sum of weighted completion times problem, since that can be easily deduced from the solution to the version with makespan constraints).

A.1 Restricted Max-Min Fair Allocation

Let us start by describing how to solve the configuration-LP for the restricted max-min fair allocation problem. Let us remind ourselves of C-LP(τ):

$$\begin{aligned} \sum_{C \in \mathscr{C}(i,\tau)} x_{iC} &\geq 1 && \forall i \in \mathscr{P} \\ \sum_{i \in \mathscr{P}} \sum_{C \in \mathscr{C}(i,\tau): j \in C} x_{iC} &\leq 1 && \forall j \in \mathscr{R} \\ 0 \leq x_{iC} &\leq 1 && \forall i \in \mathscr{P}, C \in \mathscr{C}(i,\tau) \end{aligned}$$

The first thing we have to note is that solving C-LP(τ) is non-trivial since the LP involves an exponential number of variables, and, as we will see later on, there is no trivial separation oracle for its dual. In fact, solving C-LP(τ) exactly is actually (weakly) **NP**-hard: we can straightforwardly reduce the partition Problem to solving the configuration-LP for the restricted max-min fair allocation problem. Specifically, consider an instance of the partition problem

described by a multiset of numbers S and an integer k; we create an instance of the restricted max-min fair allocation problem consisting of 2 agents i_1 and i_2, and one item of size l for each $l \in S$. One can easily see that any feasible solution to C-LP(k) for the reduced instance defines a solution to the original partition instance, since the existence of any configuration of size greater than k in the support of such a feasible solution will immediately imply that some other configuration in the support of the LP solution will have size less than k; hence, all configurations in the support of a feasible solution must have the same size k.

In spite of the above, we will be able to design a polynomial-time algorithm which returns a feasible solution to C-LP($(1-\epsilon)\tau$) if C-LP(τ) is feasible, for any $\epsilon > 0$. Specifically, let $\mathscr{C}_\epsilon(i,\tau)$ be the set of of configurations for agent i consisting of items of size at least τ/ϵ that include at most $\frac{1}{\epsilon}+1$ items; furthermore, let $\mathscr{R}^0$ be the items with size at most τ/ϵ, let $\mathscr{R}^\epsilon = \mathscr{R} \setminus \mathscr{R}^0$, let $\mathscr{R}_i$ be the set of items that can be assigned to i (analogously we define $\mathscr{R}^0_i$ and $\mathscr{R}^\epsilon_i$), and let

$$s(C) = \sum_{j \in C} p_j.$$

Now, consider the following C-LP'(τ):

$$\begin{aligned}
\sum_{C \in \mathscr{C}_\epsilon(i,\tau)} x_{iC} &\geq 1 && \forall i \in \mathscr{P} \\
x_{iC}s(C) + \sum_{j \in \mathscr{R}^0_i} x_{ijC}p_j &\geq x_{iC}\tau && \forall i \in \mathscr{P}, C \in \mathscr{C}_\epsilon(i,\tau) \\
\sum_{i \in \mathscr{P}} \sum_{C \in \mathscr{C}_\epsilon(i,\tau): j \in C} x_{iC} &\leq 1 && \forall j \in \mathscr{R}^\epsilon \\
\sum_{i \in \mathscr{P}} \sum_{C \in \mathscr{C}_\epsilon(i,\tau)} x_{ijC} &\leq 1 && \forall j \in \mathscr{R}^0 \\
x_{ijC} &\leq x_{iC} && \forall i \in \mathscr{P}, \forall j \in \mathscr{R}^0_i, C \in \mathscr{C}_\epsilon(i,\tau) \\
x_{ijC} &= 0 && \forall i \in \mathscr{P}, \forall j \notin \mathscr{R}^0_i, C \in \mathscr{C}_\epsilon(i,\tau) \\
0 \leq x_{iC} &\leq 1 && \forall i \in \mathscr{P}, C \in \mathscr{C}_\epsilon(i,\tau) \\
0 \leq x_{ijC} &\leq 1 && \forall i \in \mathscr{P}, j \in \mathscr{R}^0_i, C \in \mathscr{C}_\epsilon(i,\tau)
\end{aligned}$$

Observe that the above LP has $O(|\mathscr{P}||\mathscr{R}|^{1/\epsilon})$ variables and constraints, and therefore we can solve it in polynomial time. Furthermore, observe that any feasible solution to C-LP(τ) defines a feasible solution to C-LP'(τ). Finally, given a feasible solution to the above LP, we can transform it into a feasible solution for C-LP($(1-\epsilon)\tau$), by viewing the configurations in the support of x as machines, on which the jobs of $\mathscr{R}^0$ are fractionally assigned. Then, applying the rounding algorithm of Shmoys and Tardos [39] on these machines, will produce a fractional assignment of $\mathscr{R}^0$ to the configurations in the support of x, where all x_{ijC} are either 0 or x_{iC}, and for all $i \in \mathscr{P}$ and $C \in \mathscr{C}_\epsilon(i,\tau)$, $\sum_{j \in \mathscr{R}^0_i} x_{ijC}p_j \geq \tau - s(C) - \tau/\epsilon$ (since every job in $\mathscr{R}^0$ has size at most τ/ϵ). From this fractional assignment of items in $\mathscr{R}^0$ and the original values x_{iC} for all i and C, we can easily create a solution to C-LP$(1-\epsilon)\tau$.

A.2 Min-Sum of Weighted Completion Times with Makespan Constraints

We now present how one can approximately solve the configuration-LP required for Theorem 4.1.2. First, let us restate the LP relaxation; we call the following linear program CLP(T):

$$\begin{aligned}
\min \quad & \sum_{i \in \mathscr{M}} \sum_{C \in \mathscr{C}_i} y_{iC} \operatorname{cost}(C) \\
\text{s.t.} \quad & \sum_{C \in \mathscr{C}_i} y_{iC} \leq 1 && \forall i \in \mathscr{M}, \\
& \sum_{i \in \mathscr{M}} \sum_{C \in \mathscr{C}_i : j \in C} y_{iC} = 1 && \forall j \in \mathscr{J}, \\
& y_{iC} \geq 0 && \forall i \in \mathscr{M},\ C \in \mathscr{C}_i .
\end{aligned}$$

where $\mathscr{C}_i = \{C \subseteq \mathscr{J}_i : \sum_{j \in C} p_j \leq T\}$ is the set of jobs that can be assigned to machine i whose total size is at most T. We will state an LP-relaxation of CLP(T), and then show how we can use the relaxed LP to extract a solution whose cost is at most an $1 + O(\varepsilon)$ factor away from the optimal cost of CLP(T), and such that all the configurations fractionally assigned to machine i in the extracted solution belong to $\{C \subseteq \mathscr{J}_i : \sum_{j \in C} p_j \leq T(1+2\varepsilon)\}$.

Let us first give some intuition behind the relaxation of CLP(T): what we aim to do is replace every variable y_{iC} with a variable y_{iC_k}, where $|C| \leq \frac{1}{\varepsilon}$ (let's assume $\frac{1}{\varepsilon} \in \mathbb{N}$ for simplicity) and $k\varepsilon \operatorname{size}(C) \leq T - \sum_{j \in C} p_j$. Then, if y_{iC_k} would be set to 1 in an integral solution, it would mean that we would assign the jobs from C to i, and use a "budget" of size $k\varepsilon \operatorname{size}(C)$ to fractionally assign to i jobs of size less than $\min_{j \in C} p_j$. For each C and i, we will have a range of acceptable budgets, each of which will correspond to a variable y_{iC_k} in our linear program. Furthermore, we will have variables x_{ijC_k} that would be set to 1 if job j would be assigned to machine i by using the $k\varepsilon \operatorname{size}(C)$ budget of configuration C, and 0 otherwise; note that only a job that is smaller than all the jobs in C could be assigned to i using this budget. It is important to notice that we would include such variables only for configurations of cardinality exactly $\frac{1}{\varepsilon}$ (although it would make sense to assign some job j using the left-over budget of some configuration C where $|C| < \frac{1}{\varepsilon}$, we will always have the option of picking configuration $C \cup \{j\}$ anyway). Intuitively, C corresponds to the $\frac{1}{\varepsilon}$ largest jobs we assign to a machine, and we have an extra budget to include some smaller jobs as well; the cost induced by the jobs in C will be estimated exactly by our LP, and since the size of the rest of the jobs is relatively small, we can approximate the induced cost fairly well.

Let us now describe the relaxed LP formally. The LP will include two sets of variables. The first set will include a variable y_{iC_k} for all $i \in \mathscr{M}$, $C \subseteq \mathscr{J}_i : \sum_{j \in C} p_j \leq T \wedge |C| \leq \frac{1}{\varepsilon}$ and $k \leq |\mathscr{J}_i| :$ $k\varepsilon \operatorname{size}(C) + \sum_{j \in C} p_j < T(1+\varepsilon)$, except for those i, C, k where $|C| < \frac{1}{\varepsilon}$ and $k > 0$. The second set of variables will include a variable x_{ijC_k} for all $i \in \mathscr{M}$, $C \subseteq \mathscr{J}_i : \sum_{j \in C} p_j \leq T \wedge |C| = \frac{1}{\varepsilon}$, $k \leq |\mathscr{J}_i| :$

$k\varepsilon\,\text{size}(C) + \sum_{j\in C} p_j < T(1+\varepsilon)$ and $j \in \mathcal{J}_i : p_j \le \min_{j'\in C} p_{j'}$. If $k > 0$, we define the relaxed cost

$$\text{cost}^r(C_k) = \text{cost}(C) + (k-1)\varepsilon\,\text{size}^2(C) + \frac{((k-1)\varepsilon\,\text{size}(C))^2}{2}$$

and for $k = 0$ we define

$$\text{cost}^r(C_0) = \text{cost}(C).$$

We define

$$\mathcal{C}_i^f = \{C \subseteq \mathcal{J}_i : \sum_{j\in C} p_j \le T \wedge |C| = \frac{1}{\varepsilon}\},$$

$$\mathcal{C}_i^e = \{C \subseteq \mathcal{J}_i : \sum_{j\in C} p_j \le T \wedge |C| < \frac{1}{\varepsilon}\}$$

and

$$\mathcal{C}_i = \mathcal{C}_i^f \cup \mathcal{C}_i^e.$$

Given $C \in \mathcal{C}_i$, we define

$$\mathcal{K}_C = \begin{cases} \{0\}, \text{ if } C \in \mathcal{C}_i^e. \\ \{k \le |\mathcal{J}_i| : k\varepsilon\,\text{size}(C) + \sum_{j\in C} p_j < (1+\varepsilon)T\}, \text{ if } C \in \mathcal{C}_i^f. \end{cases}$$

Finally, we define $R(C,i) = \{j \in \mathcal{J}_i : p_j \le \min_{j'\in C} p_{j'}\}$. The relaxation of CLP($T$) is the following:

$$\begin{aligned}
\min \quad & \sum_{C\in\mathcal{C}_i}\sum_{k\in\mathcal{K}_C} y_{iC_k}\,\text{cost}^r(C_k) & & \\
\text{s.t.} \quad & \sum_{C\in\mathcal{C}_i}\sum_{k\in\mathcal{K}_C} y_{iC_k} &= 1 & \forall i \in \mathcal{M} \\
\sum_{i\in\mathcal{P}} \Big(& \sum_{C\in\mathcal{C}_i : j\in C}\sum_{k\in\mathcal{K}_C} y_{iC_k} + \sum_{C\in\mathcal{C}_i : j\in R(C,i)}\sum_{k\in\mathcal{K}_C} x_{ijC_k} \Big) &= 1 & \forall j \in \mathcal{J} \\
& x_{ijC_k} - y_{iC_k} &\le 0 & \forall i \in \mathcal{M}, C \in \mathcal{C}_i^f, k \in \mathcal{K}_C, j \in R(C,i) \\
& \sum_{j\in R(C,i)} x_{ijC_k} p_j - y_{iC_k} k\varepsilon\,\text{size}(C) &\le 0 & \forall i \in \mathcal{M}, C \in \mathcal{C}_i^f, k \in \mathcal{K}_C \\
& x_{ijC_k} &= 0 & \forall i \in \mathcal{M}, C \in \mathcal{C}_i^f, k \in \mathcal{K}_C, j \in C \\
& y_{iC_k} &\ge 0 & \forall i \in \mathcal{M}, C \in \mathcal{C}_i, k \in \mathcal{K}_C \\
& x_{ijC_k} &\ge 0 & \forall i \in \mathcal{M}, C \in \mathcal{C}_i^f, k \in \mathcal{K}_C, j \in R(C,i)
\end{aligned}$$

Observe that the relaxation contains a number of variables and constraints that is polynomial in $|\mathcal{M}|$, $|\mathcal{J}|$ and $\frac{1}{\varepsilon}$. Specifically, the relaxation includes 5 sets of constraints (excluding the non-negativity constraints). The first set of constraints states that every machine should receive one set of jobs, while the second set of constraints states that every job should be assigned to one machine. The third set of constraints states that a job can only use the budget of assigned configurations, and the fourth constraint states that the total size of jobs using a configuration's budget cannot exceed that budget. Finally, the fifth set of constraints states that a job cannot use the budget of a configuration the job already belongs to.

Next, notice that given any integral solution (y, x) to the relaxed LP, we can construct an integral

solution y' to CLP($T(1+\varepsilon)$) as follows: for each machine i, consider the unique variable y_{iC_k} that is set to 1, and the set S of jobs j such that $x_{ijc_k} = 1$. Then, set $y_{iC'} = 1$, where $C' = C \cup S$. Observe that the cost of y' can only be greater than that of (y, x).

Let us now prove that for any optimal solution (y, x) to the relaxed LP of total cost OPT, there exists a solution y^* to CLP($T(1+2\varepsilon)$), whose cost is at most $(1+O(\varepsilon))$OPT; applying Algorithm 4.1 to this solution would then imply Theorem 4.1.2, as we saw in Section 4.6.

Consider $\tau > 0$ small enough, such that y_{iC_k} is an integer multiple of τ for all $i \in \mathcal{M}$, $C \in \mathcal{C}_i$, $k \in \mathcal{K}_C$, and such that $\sum_{C \in \mathcal{C}_i : j \in R(C,i)} \sum_{k \in \mathcal{K}_C} x_{ijC_k}$ is an integer multiple of τ for all $j \in \mathcal{J}$. Given optimal solution (y, x), let us assume that for all $i \in \mathcal{M}$, $C \in \mathcal{C}_i$ and $k \in \mathcal{K}_C$, we have $y_{iC_k} \in \{0, \tau\}$, for some sufficiently small τ; this comes without loss of generality, since we could consider some C_k is assigned multiple times to i.

Now, consider some $y_{iC_k} = \tau$, where $C \in \mathcal{C}_i^f$; then,

$$y_{iC_k} k\varepsilon \operatorname{size}(C) \geq \sum_{j \in \mathcal{J}_i \setminus C} x_{ijc_k} p_j \geq y_{iC_k}(k-1)\varepsilon \operatorname{size}(C),$$

since otherwise (y, x) would not be an optimal solution. We can view x as a fractional assignment of jobs to machines, where each $j \in \mathcal{J}$ corresponds to a job, each (i, C_k) corresponds to a machine, and $\frac{x_{ijC_k}}{y_{iC_k}} = \frac{x_{ijC_k}}{\tau}$[1] is the fractional assignment of j to (i, C_k). Now, applying the algorithm of Shmoys and Tardos [39], we can extract an integral assignment of jobs to machines that correspond to some (i, C_k). More formally, we can extract an assignment x^1 such that $x^1_{ijC_k} \in \{\tau, 0\}$ and such that (y, x^1) satisfies all constraints of the relaxed LP, except for the fourth set of constraints. Regarding the fourth set of constraints, the constraint that corresponds to some $i \in \mathcal{M}, C \in \mathcal{C}_i^f, k \in \mathcal{K}_C$ might be violated by an additive $\varepsilon \operatorname{size}(C) \leq \varepsilon T$ factor, due to the application of the algorithm by Shmoys and Tardos (since C contains $\frac{1}{\varepsilon}$ jobs, for any job $j \in R(C, i)$ $p_j \leq \varepsilon \operatorname{size}(C)$).

First of all, consider any $i \in \mathcal{M}$, $C \in \mathcal{C}_i^f$, $k \in \mathcal{C}$ such that y_{iC_k} is equal to τ. From x^1, we can extract a set of jobs C' (the jobs j that have $x_{ijC_k} = \tau$), such that the total size of jobs in C' is at most $T(1+2\varepsilon)$ (an additive factor of $\varepsilon \operatorname{size}(C) \leq \varepsilon T$ comes due to (y, x^1) violating the fourth set of constraints, and an additive factor of εT comes due to the definition of $\mathcal{K}(C)$). Then, we set $y^*_{iC^*} = \tau$, where $C^* = C \cup C'$. Now, since jobs in $R(C, i)$ have size at most $\varepsilon \operatorname{size}(C)$, the cost of C^* can be upper bounded as follows:

$$\begin{aligned}
\operatorname{cost}(C^*) \leq\ & \operatorname{cost}(C) + (k+1)\varepsilon \operatorname{size}^2(C) + \tfrac{(k+1)(k+2)}{2}\varepsilon^2 \operatorname{size}^2(C) \\
\leq\ & \operatorname{cost}^r(C_k) + 2\varepsilon \operatorname{size}^2(C) + 3k\varepsilon^2 \operatorname{size}^2(C) \\
\leq\ & \operatorname{cost}^r(C_k) + 2\varepsilon \operatorname{size}^2(C) + 3\varepsilon \operatorname{size}^2(C) \\
\leq\ & \operatorname{cost}^r(C_k) + 5\varepsilon \operatorname{size}^2(C) \\
\leq\ & \operatorname{cost}^r(C_k)(1 + O(\varepsilon))
\end{aligned}$$

[1] Here we use the fact that (y, x) satisfies the third set of constraints.

since $k \leq \frac{1}{\varepsilon}$ and $\text{cost}^r(C_k) \geq \text{cost}(C) \geq \frac{\text{size}^2(C)}{2}$.

On the other hand, given any $i \in \mathscr{M}$, $C \in \mathscr{C}_i^e$ such y_{iC_k} is equal to τ, we set $y^*_{iC} = y_{iC_k}$. In this case, $\text{cost}^r(C_0) = \text{cost}(C)$. Therefore, in total the cost of solution y^* is at most $\text{OPT}(1 + O(\varepsilon))$. Finally, observe that since (y, x) satisfies the first two sets of constraints, y^* is a valid solution for $\text{CLP}(T)$.

Finally, a fine technical point is that for the above process to run in polynomial time, $1/\tau$ should be of polynomial size. We can make this assumption, by picking our original solution (y, x) to be an extreme point (which implies (y, x) contains $n^{O(\frac{1}{\varepsilon})}$ non-zero variables), and ignoring all variables in the support of (y, x) whose value is less than $\frac{1}{n^{\frac{c}{\varepsilon}}}$, for some large constant c. Then, applying Algorithm 4.1 on y^* will produce a distribution that assigns all jobs with probability at least $1 - \frac{1}{n^{\Omega(\frac{1}{\varepsilon})}}$; conditioning on the sampled solution assigning all jobs increases the cost by a multiplicative $1 + \frac{1}{n^{\Omega(\frac{1}{\varepsilon})}}$ factor.

B List of Problems

For completeness, we include a list of all the problems mentioned in the present thesis, accompanied by a formal description.

Bin Packing

- Input: A multiset $S = \{s_i : 1 \leq i \leq n\}$ of n rationals less than 1.
- Output: A number k and an assignment $f : S \to [k]$ such that $\sum_{s \in f^{-1}(i)} s \leq 1$, for all $i \in [k]$.
- Objective: Minimize k.

Generalized Assignment Problem

- Input: A set $\mathscr{P}$ of m agents, a set $\mathscr{R}$ of n items, a value $0 \leq v_{ij} \leq 1$ for each $i \in \mathscr{P}$ and $j \in \mathscr{R}$, and a weight $0 \leq w_{ij} \leq 1$ for each $i \in \mathscr{P}$ and $j \in \mathscr{R}$.
- Output: An assignment $f : \mathscr{R} \to \mathscr{P}$ such that $\sum_{j \in f^{-1}(i)} w_{ij} \leq 1$ for all $i \in \mathscr{P}$.
- Objective: Maximize $\sum_{i \in \mathscr{P}} \sum_{j \in f^{-1}(i)} v_{ij}$.

Maximum Budgeted Allocation

- Input: A set $\mathscr{P}$ of m agents, a set $\mathscr{R}$ of n items, a value $w_{ij} \in \mathbb{Q}^+$ for each $i \in \mathscr{P}$ and $j \in \mathscr{R}$, and a budget $B_i \in \mathbb{Q}^+$ for each $i \in \mathscr{P}$.
- Output: An assignment $f : \mathscr{R} \to \mathscr{P}$.
- Objective: Maximize $\sum_{i \in \mathscr{P}} \min\{B_i, \sum_{j \in f^{-1}(i)} w_{ij}\}$.

Appendix B. List of Problems

Maximum Knapsack

- Input: A multiset $W = \{w_i : 1 \leq i \leq n\}$ and a multiset $V = \{v_i : 1 \leq i \leq n\}$ of rationals less than 1.
- Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq 1$.
- Objective: Maximize $\sum_{i \in S} v_i$.

Max-Min Fair Allocation

- Input: A set $\mathscr{P}$ of m agents, a set $\mathscr{R}$ of n items, and a value $p_{ij} \in \mathbb{Q}^+$ for each $i \in \mathscr{P}$ and $j \in \mathscr{R}$.
- Output: An assignment $f : \mathscr{R} \to \mathscr{P}$.
- Objective: Maximize $\min_{i \in \mathscr{P}} \{ \sum_{j \in f^{-1}(i)} p_{ij} \}$.
- Variations: In the restricted max-min fair allocation problem, every item can be assigned to a subset of the agents, and has the same price for all of them, i.e., $p_{ij} \in \{0, p_j\}$ for all $j \in \mathscr{R}$ and $i \in \mathscr{P}$.

Minimum Knapsack

- Input: A multiset $W = \{w_i : 1 \leq i \leq n\}$ and a multiset $V = \{v_i : 1 \leq i \leq n\}$ of rationals less than 1.
- Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \geq 1$.
- Objective: Minimize $\sum_{i \in S} v_i$.

Minimum Makespan Scheduling

- Input: A set $\mathscr{M}$ of m machines, a set $\mathscr{J}$ of n jobs, and a processing time $p_{ij} \in \mathbb{Q}^+$ for each $i \in \mathscr{M}$ and $j \in \mathscr{J}$.
- Output: An assignment $f : \mathscr{J} \to \mathscr{M}$.
- Objective: Minimize $\max_{i \in \mathscr{M}} \{ \sum_{j \in f^{-1}(i)} p_{ij} \}$.

Minimum Makespan Scheduling with Costs

- Input: A set $\mathscr{M}$ of m machines, a set $\mathscr{J}$ of n jobs, a processing time $p_{ij} \in \mathbb{Q}^+$ and a cost $c_{ij} \in \mathbb{Q}^+$ for each $i \in \mathscr{M}$ and $j \in \mathscr{J}$, and a target makespan $T \in \mathbb{Q}^+$.
- Output: An assignment $f : \mathscr{J} \to \mathscr{M}$ such that $\sum_{j \in f^{-1}(i)} p_{ij} \leq T$ for all $i \in \mathscr{M}$.
- Objective: Minimize $\sum_{i \in \mathscr{M}} \sum_{j \in f^{-1}(i)} c_{ij}$.

Min Sum of Weighted Completion Times Scheduling

- Input: A set $\mathscr{M}$ of m machines, a set $\mathscr{J}$ of n jobs, a processing time p_{ij} for each $i \in \mathscr{M}$ and each $j \in \mathscr{J}$, and a weight w_j for each $j \in \mathscr{J}$.
- Output: An assignment $f : \mathscr{J} \to \mathscr{M}$ of jobs to the machines, and a schedule $\sigma_i : f^{-1}(i) \to [|f^{-1}(i)|]$ of the jobs assigned on every machine i.
- Objective: Minimize the sum of weighted completion times

$$\sum_{i \in \mathscr{M}} \sum_{j \in f^{-1}(i)} w_j \sum_{j' \in f^{-1}(i) : \sigma_i(j') \leq \sigma_i(j)} p_{ij'}.$$

- Variations:
  - In the restricted version of the problem, $p_{ij} \in \{p_j, 0\}$ for each $i \in \mathscr{M}$ and $j \in \mathscr{J}$.
  - In the uniform Smith ratios setting, we have that $p_{ij} \in \{w_j, \infty\}$ for each $i \in \mathscr{M}$ and $j \in \mathscr{J}$.

Partition

- Input: A multiset $S = \{s_i : 1 \leq i \leq n\}$ of n integers.
- Output: YES, if there exists $T \subseteq S$ such that $\sum_{s \in T} s = \frac{1}{2} \sum_{s \in S} s$, NO otherwise.

Bibliography

[1] Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. Combinatorial algorithm for restricted max-min fair allocation. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1357–1372, 2015.

[2] Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. In *APPROX–RANDOM*, pages 10–20. Springer, 2008.

[3] Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms (TALG)*, 8(3):24, 2012.

[4] Arash Asadpour and Amin Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, STOC '07, pages 114–121, New York, NY, USA, 2007. ACM.

[5] Nikhil Bansal, Aravind Srinivasan, and Ola Svensson. Lift-and-round to improve weighted completion time on unrelated machines. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 156–167, 2016.

[6] Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In Jon M. Kleinberg, editor, *STOC*, pages 31–40. ACM, 2006.

[7] Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *ACM SIGecom Exchanges*, 5(3):11–18, 2005.

[8] J. Bruno, E.G. Coffman, Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Comm. ACM*, 17:382–387, 1974.

[9] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 107–116, 2009.

[10] Deeparnab Chakrabarty and Gagan Goel. On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and gap. *SIAM J. Comput.*, 39(6):2189–2211, 2010.

[11] Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2005.

[12] F. A. Chudak. A min-sum 3/2-approximation algorithm for scheduling unrelated parallel machines. *Journal of Scheduling*, 2(2):73–77, 1999.

[13] Alan Cobham and Yehoshua Bar-Hillel. The intrinsic computational difficulty of functions. 1969.

[14] Uriel Feige. On allocations that maximize fairness. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 287–293. Society for Industrial and Applied Mathematics, 2008.

[15] Uriel Feige and Jan Vondrák. Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. In *FOCS*, pages 667–676, 2006.

[16] Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.

[17] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

[18] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1993.

[19] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the lovasz local lemma. *Journal of the ACM (JACM)*, 58(6):28, 2011.

[20] G. H. Hardy and S. Ramanujan. Asymptotic formulaæ in combinatory analysis. *Proceedings of the London Mathematical Society*, s2-17(1):75–115, 1918.

[21] Penny E Haxell. A condition for matchability in hypergraphs. *Graphs and Combinatorics*, 11(3):245–248, 1995.

[22] Han Hoogeveen, Petra Schuurman, and Gerhard J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. *INFORMS Journal on Computing*, 13(2):157–168, 2001.

[23] W.A. Horn. Minimizing average flow time with parallel machines. *Oper. Res.*, 21:846–847, 1973.

[24] Klaus Jansen and Lars Rohwedder. On the configuration-lp of the restricted assignment problem. In *SODA*, 2017, forthcoming.

[25] Christos Kalaitzis. An improved approximation guarantee for the maximum budgeted allocation problem. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1048–1066, 2016.

[26] Christos Kalaitzis, Aleksander Mądry, Alantha Newman, Lukáš Poláček, and Ola Svensson. On the configuration LP for maximum budgeted allocation. In *Integer Programming and Combinatorial Optimization - 17th International Conference, IPCO 2014, Bonn, Germany, June 23-25, 2014. Proceedings*, pages 333–344, 2014.

[27] Christos Kalaitzis, Ola Svensson, and Jakub Tarnawski. Unrelated machine scheduling of jobs with uniform smith ratios. In *SODA*, 2017, forthcoming.

[28] Tsuyoshi Kawaguchi and Seiki Kyan. Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM J. Comput.*, 15(4):1119–1129, 1986.

[29] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[30] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. A unified approach to scheduling on unrelated parallel machines. *J. ACM*, 56(5):28:1–28:31, 2009.

[31] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

[32] Lukas Polacek and Ola Svensson. Quasi-polynomial local search for restricted max-min fair allocation. In *Automata, Languages, and Programming*, pages 726–737. Springer, 2012.

[33] Thomas Rothvoß. Approximating bin packing within O(log OPT * log log OPT) bins. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 20–29. IEEE, 2013.

[34] Thomas Rothvoss. Better bin packing approximations via discrepancy theory. *SIAM J. Comput.*, 45(3):930–946, 2016.

[35] Barna Saha and Aravind Srinivasan. A new approximation technique for resource-allocation problems. In *ICS*, pages 342–357, 2010.

[36] Andreas S Schulz and Martin Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15(4):450–469, 2002.

[37] Petra Schuurman and Gerhard J Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999.

[38] Jay Sethuraman and Mark S. Squillante. Optimal scheduling of multiclass parallel machines. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland.*, pages 963–964, 1999.

[39] David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993.

[40] Martin Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *J. ACM*, 48(2):206–242, 2001.

[41] Martin Skutella and Gerhard J. Woeginger. A PTAS for minimizing the weighted sum of job completion times on parallel machines. In *Symposium on Theory of Computing, STOC*, pages 400–407, 1999.

[42] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[43] Aravind Srinivasan. Budgeted allocations in the full-information setting. In *APPROX-RANDOM*, pages 247–253, 2008.

[44] Ola Svensson. Santa claus schedules jobs on unrelated machines. *SIAM Journal on Computing*, 41(5):1318–1341, 2012.

[45] Maxim Sviridenko and Andreas Wiese. Approximating the Configuration-LP for minimizing weighted sum of completion times on unrelated machines. In *Integer Programming and Combinatorial Optimization - 16th International Conference, IPCO 2013, Valparaíso, Chile, March 18-20, 2013. Proceedings*, pages 387–398, 2013.

[46] Alan Turing. Systems of logic based on ordinals. 1938.

Christos Kalaitzis
Curriculum Vitae

École Polytechnique Fédérale de Lausanne
School of Computer and Communication Sciences
Lausanne, Switzerland

Telephone: +41 21 69 36643
E-mail: christos.kalaitzis@epfl.ch

Personal Information

- Born: November 18, 1987, in Patras, Greece.

Education

- Graduated from the 7th High School of Patras in July, 2005 (GPA: 17.9/20).
- 2005-2010: Undergraduate studies at the Department of Computer Engineering and Informatics, University of Patras.
  - Graduation: September, 2010.
  - GPA: 8.02/10.
  - Undergraduate thesis: Coordination Techniques in Strategic Congestion Games (Supervisor: Ioannis Caragiannis).
- 2010-2012: Graduate studies at the graduate program "Computer Science and Technology" at the Department of Computer Engineering and Informatics, University of Patras.
  - Graduation: July, 2012.
  - GPA: 8.56/10.
  - Graduate thesis: Efficient Addressing and Routing in Large-scale Communication Networks (Supervisor: Ioannis Caragiannis).
- 2012-today: Doctoral studies at the Theory of Computation Laboratory of the School of Computer and Communication Sciences, EPFL, under the supervision of prof. Ola Svensson.
  - Graduation (expected): February/March, 2017.
  - Doctoral Dissertation: Applications of Strong Convex Relaxations to Allocation Problems (Advisor: Prof. Ola Svensson).

Research Interests

- Approximation Algorithms
- Computational Complexity
- Algorithmic Game Theory

Conference Publications

- C. Kalaitzis, O. Svensson and J. Tarnawski. Unrelated Machine Scheduling of Jobs with Uniform Smith Ratios. In *ACM-SIAM Symposium on Discrete Algorithms (SODA) 2017*, forthcoming.
- C. Kalaitzis. An Improved Approximation Guarantee for the Maximum Budgeted Allocation Problem. In *ACM-SIAM Symposium on Discrete Algorithms (SODA) 2016*, pp. 1048-1066, 2016.
- C. Annamalai, C. Kalaitzis and O. Svensson. Combinatorial Algorithm for Restricted Max-Min Fair Allocation. In *ACM-SIAM Symposium on Discrete Algorithms (SODA) 2015*, pp. 1357-1372, 2015.
- C. Kalaitzis, A. Mądry, A. Newman, L. Poláček and O. Svensson. On the Configuration LP for Maximum Budgeted Allocation. In *17th Conference on Integer Programming and Combinatorial Optimization (IPCO 2014)*, pp. 333-344, 2014.
- J. Augustine, I. Caragiannis, A. Fanelli, and C. Kalaitzis. Enforcing efficient equilibria in network design games via subsidies. In *24th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2012)*, pp. 277-286, 2012.
- I. Caragiannis and C. Kalaitzis. Space lower bounds for low-stretch greedy embeddings. In *19th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2012)*, pp. 1-12, 2012.

Journal Publications

- I. Caragiannis and C. Kalaitzis. Space lower bounds for low-stretch greedy embeddings. In *Theoretical Computer Science*, 610, pp. 149-157, 2016.
- C. Kalaitzis, A. Mądry, A. Newman, L. Poláček and O. Svensson. On the Configuration LP for Maximum Budgeted Allocation. In *Mathematical Programming*, 154(1-2), pp. 427-462, 2015.
- J. Augustine, I. Caragiannis, A. Fanelli, and C. Kalaitzis. Enforcing efficient equilibria in network design games via subsidies. In *Algorithmica*, 72(1), pp. 44-82, 2015.

Teaching

- Spring 2013: Teaching assistant for the course: Advanced Theoretical Computer Science.
- Fall 2013: Teaching assistant for the course: Algorithms (Outstanding TA award).
- Spring 2014: Teaching assistant for the course: Topics in Theoretical Computer Science.
- Fall 2014: Teaching assistant for the course: Algorithms.
- Spring 2015: Teaching assistant for the course: Topics in Theoretical Computer Science.
- Fall 2015: Teaching assistant for the course: Algorithms.
- Spring 2016: Teaching assistant for the course: Theory of Computation.