PARAMETERIZED ALGORITHMS FOR INTEGER LINEAR PROGRAMS AND THEIR APPLICATIONS FOR ALLOCATION PROBLEMS

ALEXANDRA ANNA LASSOTA

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
der Technische Fakultät der Christian-Albrechts-Universität zu Kiel

März 2021

1. GUTACHTER/IN Prof. Dr. Klaus Jansen

2. GUTACHTER/IN Prof. Dr. Nicole Megow

3. GUTACHTER/IN Prof. Dr. Friedrich Eisenbrand

DATUM DER DISPUTATION 18.06.2021

Alexandra Anna Lassota: *Parameterized Algorithms for Integer Linear Programs and Their Applications for Allocation Problems,* DISSERTATION, © März 2021

Dedicated to
my mother and father

ABSTRACT

This thesis is concerned with solving NP-hard problems. We consider two prominent strategies of coping with such computationally hard questions efficiently. The first approach aims to design approximation algorithms, that is, we are content to find good, but non-optimal solutions in polynomial time. The second strategy is called Fixed-Parameter Tractability (FPT) and considers parameters of the instance to capture the hardness of the problem and by that, obtain efficient algorithms with respect to the remaining input. This thesis employs both strategies jointly to develop efficient approximation and exact algorithms using parameterization and modeling the problem as structured integer linear programs (ILPs), which can be solved in FPT. In the first part of this work, we concentrate on these well-structured ILPs. On the one hand, we develop an efficient algorithm for block-structured integer linear programs called n-fold ILPs. On the other hand, we investigate the similarly block-structured 2-stage stochastic ILPs and prove conditional lower bounds regarding the running time of any algorithm solving them that match the best known upper bounds. We also prove the tightness of certain structural parameters called sensitivity and proximity for ILPs which arise from combinatorial questions such as allocation problems. The second part utilizes n-fold ILPs and structural properties to add to and improve upon known results for Scheduling and Bin Packing problems. We design exact FPT algorithms for the Scheduling With Clique Incompatibilities, Bin Packing, and Multiple Knapsack problems. Further, we provide constant-factor approximation algorithms and polynomial time approximation schemes (PTAS) for the Class Constraint Scheduling problems. Broadening our scope, we also investigate this problem and the closely related Cardinality Constraint Scheduling problem in the online setting and derive lower bounds for the approximation ratios as well as a PTAS for them. Altogether, this thesis contributes to the knowledge about structured ILPs, proves their limits and reaffirms their usefulness for a plethora of allocation problems. In doing so, various new and improved algorithms with respect to the running time or approximation quality emerge.

ZUSAMMENFASSUNG

Diese Arbeit beschäftigt sich mit dem Lösen NP-schwerer Probleme. Um diese Probleme adäquat anzugehen, werden wir uns zweier prominenter Strategien bedienen. Der erste Ansatz befasst sich mit dem Entwurf von Approximationsalgorithmen, wir begnügen uns also mit guten, aber nicht optimalen Lösungen, die in polynomieller Zeit berechenbar sind. Die zweite Strategie, *Fixed-Parameter Tractability* (FPT) genannt, berücksichtigt zusätzlich Parameter der Instanz, die die Schwierigkeit des Problems einfangen und dadurch das Entwickeln effizienter Algorithmen bezüglich der restlichen Eingabe ermöglichen. Diese Arbeit wendet beide Strategien gemeinsam an und entwickelt so effiziente Approximations- und exakte Algorithmen unter Verwendung der Parametrisierung und Modellierung der Probleme als strukturierte ganzzahlige lineare Programme (ILPs), die in FPT gelöst werden können. Im ersten Teil dieser Arbeit konzentrieren wir uns auf diese wohlstrukturierten ILPs. Zum einen entwickeln wir einen effizienten Algorithmus für blockstrukturierte ganzzahlige lineare Programme, die n-fold ILPs genannt werden. Andererseits untersuchen wir die ähnlich blockstrukturierten 2-stage stochastic ILPs und beweisen für jeden Algorithmus zum Lösen dieser ILPs untere Laufzeitschranken, welche mit den bestbekannten Algorithmen übereinstimmen. Des Weiteren beweisen wir, dass die Abschätzungen bestimmter struktureller Parameter, genannt *sensitivity* und *proximity*, auch nicht für ILPs verbessert werden können, die aus kombinatorischen Fragestellungen wie beispielsweise aus Packungsproblemen entstehen. Der zweite Teil verwendet n-fold ILPs und strukturelle Eigenschaften, um bekannte Ergebnisse für Scheduling und Bin Packing Probleme zu ergänzen und zu verbessern. Wir entwerfen exakte FPT-Algorithmen für *Scheduling With Clique Incompatibilities*, *Bin Packing*, und *Multiple Knapsack*. Außerdem präsentieren wir Approximationsalgorithmen mit konstanter Approximationsrate und Polynomialzeit-Approximationsschemata (PTAS) für *Class Constraint Scheduling* Probleme. Darüber hinaus untersuchen wir diese Probleme und das eng verwandte *Cardinality Constraint Scheduling* Problem aus der Perspektive von Online-Algorithmen und leiten untere Schranken bezüglich der Approximierbarkeit, sowie ein PTAS für sie ab. Insgesamt trägt diese Arbeit zum Wissen über strukturierte ILPs bei, zeigt deren Grenzen auf und bestätigt ihre Nützlichkeit für viele Zuweisungsprobleme. Dabei entstehen neue und verbesserte Algorithmen bezüglich der Laufzeit oder der Approximationsqualität.

ACKNOWLEDGMENTS

First of all, I sincerely thank my adviser Klaus Jansen, who gave me the opportunity to join his amazing group, introduced me to a plethora of interesting questions, and supported me throughout my studies.

Further, I am truly grateful for Sebastian Berndt, who beside being an excellent co-worker and co-author, also became a great friend and the best confident for all concerns. I wish you and your family all the best.

Special thanks goes out to Thomas Wilke for hosting me during the last months of my studies.

I also thank all my other co-workers, co-authors and collaborators Max Bannach, Cornelius Brand, Hauke Brinkop, Max Deppert, Leah Epstein, Katrin Flöth, Kilian Grage, Ute Iaquinto, Kim-Manuel Klein, Leonie Krull, Asaf Levin, Marten Maack, Parvaneh Massouleh, Matthias Mnich, Tytus Pikies, Malin Rau, Lars Rohwedder, Henrik Schmidt, Malte Skambath, and Malte Tutas. It was always an enormous pleasure working, traveling and puzzling over various questions with you.

Last, but not least, I heartily thank my family for their everlasting support and Dennis Last for always being there for me and keeping up with me.

CONTENTS

Home is behind, the world ahead,
and there are many paths to tread
through shadows to the edge of night,
until the stars are all alight.

— from *The Lord of the Rings* by J. R. R. Tolkien

1 INTRODUCTION

Computer science is about solving problems. Some of the most interesting and intensively studied ones are the problems which are considered *hard*. In our understanding, hard means that we are currently not able to solve these problems *efficiently*. We even suspect that this is not possible at all. In our context, efficiency is measured by the time needed to solve a problem with respect to the size of the problem instance. An algorithm is called efficient if its running time is bounded by a polynomial in the size of the problem instance with constant exponents.

As an example, let us consider the classical Scheduling problem. An instance of this problem consists of a set of jobs and a set of machines. Further, each job has some processing time, i. e., the duration required to execute the job completely. A *schedule* is an assignment of all jobs onto the given machines. The quality of a schedule is measured by the maximal sum of processing times on a machine, called *makespan*. The goal is to find a schedule that minimizes the makespan. A simple solution would assign all jobs onto the first machine. In most cases, the resulting makespan is a lot larger than in an optimal solution. More cleverly, we can consider the jobs in descending order by their processing times and distribute them onto the machines which are considered in some fixed order which is repeated after each machine received a job. This algorithm is called *round robin*. Round robin as well as the aforementioned algorithm are both efficient regarding the computation time, but not optimal regarding the quality of the solution. If we aim for an optimal solution, the best known algorithm of the last decades, neglecting some nonessential improvements, is to test all possible assignments. If we have N jobs and M machines, this yields M^N guesses, which bounds the running time from beneath. Thus, this algorithm is not efficient, as the number N appears in the encoding length of the instance. Further, already for relatively small instances, this is simply not computable in reasonable time. Nonetheless, if we are given an assignment, it is easy to verify that the solution is a valid schedule, i. e., all jobs are scheduled onto the given set of machines.

For technical reasons, let us consider the underlying decision variant of the above Scheduling problem, which asks whether there exists a solution satisfying a given makespan. Note that deciding such a problem is as most as hard as finding a solution. We call such decision problems *NP-complete*

which share the trait that deciding them is hard, but verifying the feasibility of a potential solution can be done efficiently. If we are faced with one of those problems, we have some strategies to deal with them satisfactorily. In this thesis, we employ two prominent ones, in particular, approximation algorithms and Fixed-Parameter Tractability.

The idea of approximation algorithms is already used above: Instead of computing an optimal solution to an optimization problem, we are satisfied to produce one of sufficient quality efficiently. For example, the *round robin* algorithm computes a 2-approximation for the Scheduling problem, i. e., it takes at most twice as long for all jobs to be finished than it would in an optimal schedule. Sometimes, we can even compute solutions whose quality comes arbitrarily close to the optimum. This means, for any fixed $\epsilon > 0$, we can design an algorithm for a minimization problem that computes a solution in polynomial time which is only worse by a factor of $(1 + \epsilon)$ regarding an optimal solution. For maximization problems, the quality of the solution is worse by a factor of $(1 - \epsilon)$. Such algorithms form a family, which is called *Polynomial Time Approximation Schemes*, or PTAS for short. Already for decades, such approximation algorithms are intensively studied. The history and the manifoldness of results are thus too large to adequately present them here. We refer to the books [161, 164] and surveys [29, 151] for a proper covering of this topic. Detailed portrayals of the related work for the problems studied throughout this thesis are presented in the respective chapter.

Nonetheless, not all problems can be approximated efficiently. Further, problems that are not optimization ones cannot be tackled by this approach. In these cases, it may be reasonable to examine properties about the structure of the input instance such as the number of different processing times. This additional information may for example be derived from the type of application. Such properties may then become the key elements for designing a faster algorithm which computes an optimal solution. The research direction that considers further information about the problem or parameters of the instance to obtain faster algorithms is called Fixed-Parameter Tractability, or FPT for short. A (decision) problem is in the complexity class FPT if an algorithm solving it runs in time $\text{poly}(|I|) \cdot f(k_1, k_2, \dots)$, i. e., polynomial in the size of the instance $|I|$ and with some dependency on the additional parameters $k_1, k_2, \dots$ (if existent). For a detailed presentation on the history of this field and an overview on the results, we refer to the books [41, 44, 61]. Let us consider the Bin Packing problem as an example of successfully utilizing structural properties. In this problem, we are given N items with certain sizes. The goal is to pack them into as few unit sized bins as possible. The decision variant of this problem, asking whether some number of bins is sufficient, is NP-complete. However, if we parameterize the problem by the structural property that at most two items fit into a bin without exceeding its capacity, the problem becomes efficiently solvable: formulate it as a matching problem, solve the matching problem optimally, and compare the size of the matching to the desired number of bins. Roughly speaking, the items become

nodes and two nodes are connected by an edge if they fit together into a bin. A maximum matching, i. e., the largest possible subset of the edges where no two edges connect with the same node, directly translates to an optimal packing: For each edge, pack the items represented by its nodes together into a bin. Nodes that are not matched are packed separately. Thus, the decision variant of the Bin Packing problem with the structural property that at most two items fit together into a bin and no further parameters is efficiently solvable.

Both approaches may provide the desired solutions to our hard problems. But if this is not the case, or if we want to improve upon the algorithms with respect to the running time or quality of a solution, we can go even further: It is possible to employ both ideas jointly and this thesis explores the possibilities and limits of doing so. There is a successful line of results confirming the fruitfulness of combining these approaches, for an overview we refer to the surveys [130, 133, 134]. The central theme we explore in the following is to model allocation problems as *integer linear programs*, or ILPs for short. These programs are a powerful tool in the design of various algorithms, as it allows us to express a plethora of linear goals (like makespan minimization as the sum of the allocated processing times) while satisfying different conditions (like scheduling all jobs onto the given set of machines under certain restrictions). These conditions are expressed as the rows of a matrix, called *constraint matrix*, and each column corresponds to a part of the desired solution. In general, solving integer linear programs is also hard. Thus, we investigate special block-structures of the constraint matrices such that the underlying ILPs can be solved efficiently, yet are general enough to model our allocation problems. In particular, we investigate ILPs in which the variables only interact locally with few other variables, but are tied together by few global constraints, or by duality, are tied together only by interacting with a fixed, small set of variables. To model the respective allocation problem appropriately as such block-structured ILPs, we have to simplify our allocation problems slightly. We preprocess the problem for example by rounding the processing times and items sizes respectively. Therefore, we get an approximation algorithm. We show throughout this work that this combination of two approaches yields new, more efficient and, regarding the quality of the solution, better algorithms. But, as always, there are also limits to this approach and we explore them by proving lower bounds for the structured ILPs.

STRUCTURE OF THE THESIS The next chapter introduces the basic concepts and definitions used throughout this work. The remainder of the thesis is divided into two parts.

Part I, called *Integer Linear Programs*, investigates the possibilities and limits of block-structured ILPs by designing efficient algorithms and proving lower bounds regarding their running times. Further, we show the tightness of known estimations regarding the size of the structural properties called *proximity* and *sensitivity* for ILPs which arise from allocation problems.

In Part II, called *Allocation Problems*, we explore the approach to model our respective allocation problems as integer linear programs or use their structural properties to solve them efficiently and with the desired quality of the solution. By that, we design various new and efficient algorithms such as polynomial time approximation schemes (PTASs) or exact FPT algorithms, which improve upon the previous results known about them.

Each chapter is structured as follows: First, we familiarize ourselves with the problem; then we list all results presented in that chapter; this is followed by a detailed related work section for that particular problem; then we explain the structure of the remainder of the chapter; and finally, prove the results.

Each part is concluded with an open questions chapter, which emphasizes and explains the most interesting open problems and possible further research directions related to the established results.

Most of the presented results are based on the following publications.

PUBLICATIONS

[14] M. Bannach, S. Berndt, M. Maack, M. Mnich, A. Lassota, M. Rau, and M. Skambath. "Solving Packing Problems with Few Small Items Using Rainbow Matchings." In: *MFCS*. Vol. 170. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 11:1–11:14.

[19] S. Berndt, K. Jansen, and A. Lassota. "Tightness of Sensitivity and Proximity Bounds for Integer Linear Programs." In: *SOFSEM*. Vol. 12607. Lecture Notes in Computer Science. Springer, 2021, pp. 349–360.

[96] K. Jansen, K. Klein, and A. Lassota. "The Double Exponential Runtime is Tight for 2-Stage Stochastic ILPs." In: *IPCO*. Lecture Notes in Computer Science. to appear. Springer, 2021.

[100] K. Jansen, A. Lassota, and M. Maack. "Approximation Algorithms for Scheduling with Class Constraints." In: *SPAA*. ACM, 2020, pp. 349–357.

[101] K. Jansen, A. Lassota, M. Maack, and T. Pikies. "Total Completion Time Minimization for Scheduling with Incompatibility Cliques." In: *ICAPS*. to appear. AAAI Press, 2021.

[102] K. Jansen, A. Lassota, and L. Rohwedder. "Near-Linear Time Algorithm for n-fold ILPs via Color Coding." In: *SIAM Journal on Discrete Mathematics* 34.4 (2020). An extended abstract of this work appeared in the proceedings of the 46th International Colloquium on Automata, Languages and Programming (ICALP 2019), pp. 2282–2299.

DISCUSSION OF THE RESULTS

In the first part of this thesis, called *Integer Linear Programs*, we investigate upper and lower bounds regarding algorithms and structural properties for well-structured ILPs.

CHAPTER 3: NEAR-LINEAR TIME ALGORITHM FOR n-FOLD ILPS We start with studying n-fold integer linear programs. These block-structured ILPs have a constraint matrix where non-zero entries only appear in the first few rows and block-wise along the diagonal beneath. These ILPs are in FPT parameterized by the block dimensions and the largest entry of the constraint matrix. We present an algorithm for solving n-fold ILPs optimally in time near-linear in the number of columns (and thus blocks) and single-exponentially in the block dimensions and the largest entry of the constraint matrix. By that, we improve upon the previous best running times with respect to the dependency on the number of columns and the block dimensions. Further, in contrast to earlier work, we manage to circumvent the necessity to compute the LP relaxation to give artificial upper bounds on unbounded variables. The basic idea of the algorithm is to compute some initial solution and improve it step-wise to optimality. The main improvement with respect to previous algorithms is to set up and maintain a data structure over all iterations to speed up the computation of the augmentation steps. The data structure is based on color coding, a powerful tool which is used extensively in FPT algorithms, but was not applied in this context before. Finally, we also present new structural results for n-fold ILPs regarding the distance between two optimal solutions with respect to different right-hand sides, called *sensitivity*. We prove that it only depends on the change of the right-hand side, the block dimensions and the largest entry of the constraint matrix. It is thus independent of the overall number of columns or rows as for general ILPs. These results are also presented in [102].

CHAPTER 4: HARDNESS FOR 2-STAGE STOCHASTIC ILPS This chapter investigates the transpose of the n-fold ILPs, called 2-stage stochastic. In contrast to n-fold ILPs, no algorithm with a single exponential dependency on the block dimension is known for them. We prove that 2-stage stochastic integer linear programs are indeed intrinsically harder to solve. We do so by showing a reduction from the 3-SAT problem to a decision variant of 2-stage stochastic ILPs. Assuming the exponential time hypothesis (ETH), a widely believed conjecture stating an exponential lower bound on the running time of any algorithm solving the 3-SAT problem, we derive a double exponential dependency on the block dimensions for any algorithm solving 2-stage stochastic integer linear programs. This bound (nearly) matches the current state-of-the-art algorithms. The key part of the reduction lies in the transformation of a 3-SAT instance to an equivalent instance of the Quadratic Congruences problem, a number theoretical problem asking whether there exists some small number satisfying a quadratic equation modulo a given number. A similar reduction was already known, but the parameter dependencies were too large to derive our desired lower bounds. Thus, we give a new reduction which heavily relies on prime numbers and the properties of their products. These results can also be found in [96].

CHAPTER 5: TIGHTNESS OF SENSITIVITY AND PROXIMITY BOUNDS Here, we investigate the well-known bounds shown by Cook et al. on the *proximity*, i. e., the maximal distance between any optimal fractional solution and its closest integral one, and the *sensitivity* of general ILPs. It is known that these bounds are tight. However, all such examples either only have small values in the constraint matrix, use negative entries in that matrix, or have non-integral right-hand sides. Thus, the examples do not correspond to instances from algorithmic questions such as the aforementioned SCHEDULING or BIN PACKING problems. We show that both structural properties are indeed tight by giving appropriate examples for arbitrarily large, integral and positive numbers in the ILP. These examples can further be extended to correspond to instances from the BIN PACKING problem. The results are based on [19].

The second part of this thesis, called *Allocation Problems*, investigates various scheduling and bin packing problems and some of their natural variants. We add to the knowledge of these problems and improve upon previous results by utilizing different integer linear programming techniques.

CHAPTER 7: SCHEDULING WITH CLIQUE INCOMPATIBILITIES The second part starts with the SCHEDULING WITH CLIQUE INCOMPATIBILITIES problem with respect to the minimization of the sum of (weighted) completion times, one of the most studied objectives in scheduling. In addition to the classical SCHEDULING problem, we are given a disjoint partition of the jobs and no two jobs from the same partition are allowed to be scheduled onto the same machine. Representing this as a graph where each node corresponds to a job and an edge represents a conflict, we get a graph consisting of disjoint cliques, hence the name. We tackle this problem from an FPT point of view: We investigate three natural parameterizations and prove that the corresponding problems are in FPT. In particular, we parameterize by combinations of number of cliques, number of machines, largest processing time, and number of job kinds. In this context, jobs are of the same *kind* if they are indistinguishable. The algorithmic approach explores and reaffirms the applicability of n-fold ILPs, as the fundamental idea for each parameterization is to formulate the problem as a *configuration ILP* with that specific structure and block dimensions which are only dependent on a function of the respective parameters. Thus, they can be solved efficiently as shown in Part I. Roughly speaking, a configuration ILP assigns a subset of jobs, called *configuration*, onto each machine such that all jobs are covered and each machine only gets one feasible configuration. This solution can then be turned into a feasible, optimal schedule with a minimal sum of (weighted) completion times. This chapter is part of the paper [101].

CHAPTER 8: CLASS CONSTRAINT SCHEDULING: CONSTANT-FACTOR APPROXIMATION The CLASS CONSTRAINT SCHEDULING problem is a scheduling variant where each job additionally has some class and each machine can

only schedule a limited number of different classes. The limit is equal on each machine. The objective is to minimize the makespan while not exceeding the class limitations on the machines. This chapter is concerned with simple and efficient constant-factor approximation algorithms for the different variants of feasibly allotting the jobs onto the machines, i. e., whether preemption or a parallel execution of pieces from the same job are allowed. These variants are called *non-preemptive*, *preemptive* and *splittable*. We establish a framework and give explicit algorithms for each variant yielding a $7/3$-approximation for the non-preemptive case and 2-approximation algorithms for the other cases respectively. These results are part of the paper [100].

CHAPTER 9: VARIABLE CLASS CONSTRAINT SCHEDULING: PTAS This chapter adds to the results of the previous one. Here, we consider the VARIABLE CLASS CONSTRAINT SCHEDULING problem, where, in contrast to before, the limits for the machines can differ. We assume that the number of different values is a parameter. We present a PTAS for each of the allocation variants. Following the central theme of this thesis, the key element is to formulate the respective problem as an n-fold ILP. However, due to the complexity of the problem, involved pre- and postprocessing steps are necessary. Among others, we have to appropriately group the jobs from the same classes and prove structural properties of (nearly) optimal solutions. Only then is it possible to set up the desired ILPs where the number of variables and constraints in a block only depend on a function of the parameters and by that, to derive a feasible and nearly optimal solution efficiently. This chapter extends our results from [100] where the class limits are identical for all machines.

CHAPTER 10: SOLVING PACKING PROBLEMS WITH FEW SMALL ITEMS We already discussed that structural properties or further knowledge about a (hard) problem may be the key element for designing exact and efficient algorithms. This chapter studies the vector variants of the BIN PACKING and MULTIPLE KNAPSACK problem, i. e., where the items and the bins have multi-dimensional sizes that are represented by a vector. We parameterize by the number of small items with respect to a suitable and natural definition of smallness. Roughly speaking, a multidimensional item is small if it fits into a multidimensional bin together with two additional large items. We give a randomized FPT algorithm (with one-sided error) for each problem. The most interesting part of the algorithms is to transform the respective instance to a generalization of the matching problem. Further, to avoid randomization introduced while finding the aforementioned matching, we also give a deterministic FPT algorithm for the one-dimensional BIN PACKING problem, which we set up by proving various structural properties of optimal solutions. These results can also be found in [14].

CHAPTER 11: ONLINE CARDINALITY CONSTRAINT SCHEDULING In the final chapter of this thesis, we investigate the SCHEDULING problem from another point of view: For some applications, it is too restrictive to assume

that the whole input is known from the beginning or that the instance is static. Regarding the SCHEDULING problem, some jobs might arrive over time. To handle this, online algorithms are necessary, which can operate on changing instances. The goal is to design an algorithm, which produces a good solution efficiently while altering the current solution after each change of the instance as little as possible. Obviously, there is a trade-off between running time, quality of a solution and amount of allowed changes. In this chapter, we study the case that no changes to the already placed jobs are allowed or we can re-schedule jobs with a total processing time dependent on the processing time of the new job. For both cases, we prove that in the worst case, any algorithm for the CLASS CONSTRAINT SCHEDULING problem produces a solution with a quality M times larger than an optimal one, where M is the number of machines in the instance. Thus, this problem seems rather hopeless in the online setting. To gain deeper understanding on the complexity of this problem, we also investigate an important subcase, called CARDINALITY CONSTRAINT SCHEDULING. In this problem, each job has unique class, i. e., the class restriction is equivalent to a restriction on the number of jobs that a machine can schedule. If no changes are allowed, we prove a lower bound of nearly 2 regarding the quality of a solution of any online algorithm compared to an optimal offline algorithm. If, however, we allow changes dependent on the size of the job limitation of a machine, some accuracy factor and the processing time of the new job, we design a PTAS. The idea is to set up a configuration ILP and use sensitivity results. By that, we only alter the current optimal solution by changing few configurations and thus, only few machines and their corresponding jobs. Preliminary versions of these and further results were discussed with Leah Epstein, Klaus Jansen, Asaf Levin, Marten Maack and Lars Rohwedder and are part of the unpublished joint work [55].

You're saying it wrong. It's Wing-gar-dium Levi-o-sa, make the 'gar' nice and long.

— from *Harry Potter and the Sorcerer's Stone* by J. K. Rowling

2 PRELIMINARIES

This chapter introduces the basic concepts and definitions necessary and repeatedly used throughout this work. Further, the main results for some of the problems are presented if applied in the following chapters.

POLYNOMIAL ALGORITHMS We categorize algorithms with respect to their running time as follows. If an algorithm runs in polynomial time with respect to the encoding length of the input, we call it *polynomial*. If the running time depends polynomially on the value of the numerical inputs, we say that the algorithm runs in *pseudo-polynomial* time. Polynomial algorithms can be further divided into *strongly polynomial* and *weakly polynomial*. This is in reference to the so-called arithmetic model of computation, where all arithmetic on the numeric inputs can be performed in unit time. We call a polynomial algorithm strongly polynomial if its running time in this model can be bounded by a polynomial that depends only on the size of the numbers (but not the values), and weakly polynomial otherwise.

FIXED-PARAMETER TRACTABILITY A *parameterized problem* is a subset $L \subseteq \{0,1\}^* \times \mathbb{N}$ called *language*. The first entry encodes the instance and the second one, called *parameter*, some further knowledge about the problem. For example, the parameter could be the treewidth of the instance or the size of a solution. Such a problem is *fixed-parameter tractable* (FPT) if there is an algorithm that decides if $(I,k) \in L$ in time $f(k) \cdot \text{poly}(|I|)$ where f is some computable function and $|I|$ denotes the encoding length of I. This definition can easily be extended to multiple parameters, for example, by summing them up. We also talk about FPT algorithms in the context of optimization problems. In that case, we say that a problem is FPT if it can be solved optimally in time $f(k) \cdot \text{poly}(|I|)$ for a computable function f. A *parameterized reduction* from a parameterized problem L to another problem L' is an algorithm that transforms an instance (I,k) into (I',k') such that

- $(I,k) \in L \Leftrightarrow (I',k') \in L'$,
- $k' \leq f(k)$, and
- it runs in time $f(k) \cdot \text{poly}(|I|)$.

SCHEDULING AND PACKING PROBLEMS Scheduling and packing problems are fundamental in theoretical computer science and operations research.

The objective is to place N objects into (bounded or arbitrarily many) locations such that some objective is minimized. In the following, we present two basic variants.

In the classical SCHEDULING problem, the objects are called *jobs* and the locations are called *machines*. In particular, we are given a set $\mathcal{J}$ of N jobs with processing times $p_1, \dots, p_N$ and a set $\mathcal{M}$ of M machines. The objective is to find a placement of the jobs onto the machines, called *schedule*, that minimizes the *makespan*. The makespan is the maximal sum of processing times on a machine. We differentiate between three ways of feasibly placing the jobs, i. e., whether preemption of a job is allowed and whether the job pieces of the same job can be scheduled in parallel. Formally, we define a schedule σ and the corresponding makespan $\mu(\sigma)$ as follows for each case:

- *Non-preemptive case:* In this case, the jobs are placed as a whole, thus the schedule is most straight-forward to define. A schedule $\sigma\colon \mathcal{J} \to \mathcal{M}$ assigns each job $j \in \mathcal{J}$ onto a machine $\sigma(j) = i \in \mathcal{M}$. The makespan $\mu(\sigma)$ of a schedule σ is defined by the maximum sum of processing times a machine has to schedule, i. e., $\mu(\sigma) = \max_{i\in\mathcal{M}}\{\sum_{j\in\sigma^{-1}(i)} p_j\}$. An exemplary schedule is presented in Figure 2.1.

- *Splittable case:* In the splittable case, we are allowed to cut the jobs into arbitrary small pieces and place them anywhere as long as we do not schedule two jobs simultaneously on the same machine. Formally, we define a function $\omega\colon \mathcal{J} \to \mathbb{Z}_{\geq 0}$ which maps every job to the number of pieces it is split into. Further, we define a function $\lambda_j\colon [\omega(j)] \to (0,1]$ such that $\sum_{k\in\omega(j)} \lambda_j(k) = 1$ which states the fraction of the overall processing time of job j for each part. Define $\mathcal{J}' = \{(j,p) \mid j \in \mathcal{J}, p \in [\omega(j)]\}$ as the set of job parts. An assignment $\psi\colon \mathcal{J}' \to \mathcal{M}$ matches job pieces to machines. Finally, we define a schedule $\sigma = (\omega, \lambda, \psi)$. The makespan $\mu(\sigma)$ of a schedule σ is defined by the maximum sum of processing times on a machine, i. e., $\max_{i\in\mathcal{M}}\{\sum_{(j,p)\in\psi^{-1}(i)} \lambda_j(p)p_j\}$.

- *Preemptive case:* This case resembles the splittable case, but in addition, pieces of the same job are not allowed to be scheduled in parallel, i. e., at the same time on different machines. Consider the definitions as in the splittable case. However, we cannot solely assign job pieces to machines as before. We need to state the starting points of the pieces additionally. Let $\xi\colon \mathcal{J}' \to \mathbb{Q}_{\geq 0}$ be a function defining the starting time of a job piece. A schedule is defined as $\sigma = (\omega, \lambda, \xi, \psi)$ and it has to hold that for each two job parts $(j,p), (j',p') \in \mathcal{J}'$ with $\psi(j,p) = \psi(j',p')$ or $j = j'$ that $\xi(j,p) + \lambda_j(p)p_j \leq \xi(j',p')$ or $\xi(j',p') + \lambda_{j'}(p')p_{j'} \leq \xi(j,p)$. In other words, job pieces on the same machine or pieces belonging to the same job are not allowed to be scheduled in parallel, i. e., not at the same time. The makespan is again given by $\max_{i\in\mathcal{M}}\{\sum_{(j,p)\in\psi^{-1}(i)} \lambda_j(p)p_j\}$.

Let us turn our attention to another prominent problem: In the classical BIN PACKING problem, the objects are called *items* and the locations are called *bins*.

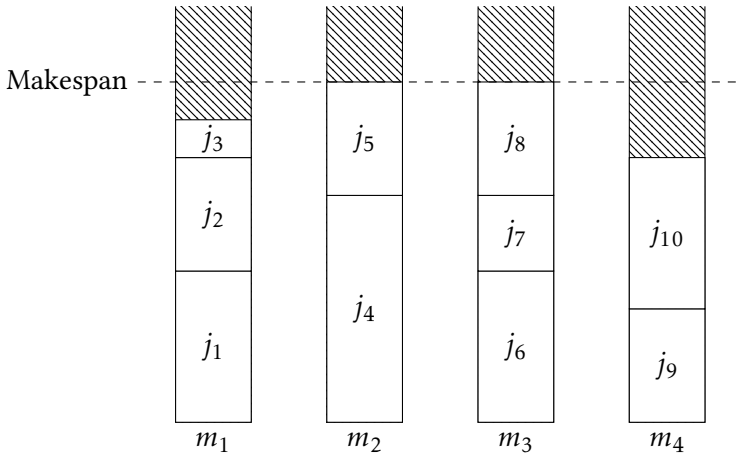

Figure 2.1: An exemplary, optimal and non-preemptive schedule for 10 jobs and 4 machines

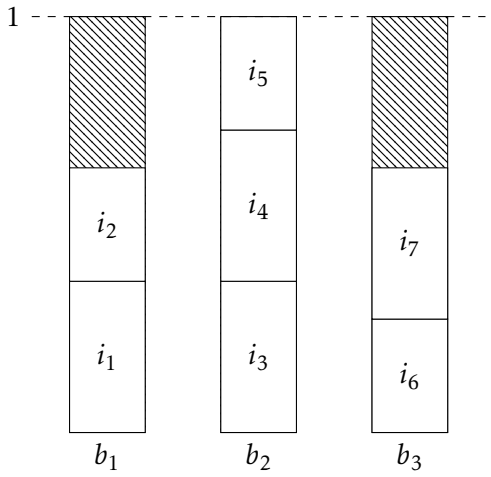

Figure 2.2: An exemplary, optimal packing for 7 items

In particular, we are given a set $\mathcal{I}$ of N items with sizes $s_1, \ldots, s_N \in (0,1]$. The objective is to place all items into as few unit sized bins as possible. Formally, we aim to find a *packing* $\rho\colon \mathcal{I} \to \mathbb{N}$ such that $\sum_{j \in \rho^{-1}(i)} s_j \leq 1$ holds for all bins $i = 1, 2, \ldots$ and which minimizes $\max\{\rho(j) \mid j \in \mathcal{I}\}$. An exemplary packing is presented in Figure 2.2.

BLOCK-STRUCTURED INTEGER LINEAR PROGRAMS *Integer linear programs* (ILPs) are a powerful and omnipresent tool to model various (optimization) problems such as the aforementioned scheduling and packing ones. In general, we are given some matrix $A \in \mathbb{Z}^{d_1 \times d_2}$, a right-hand side $b \in \mathbb{Z}^{d_1}$ and a cost vector $w \in \mathbb{Z}^{d_2}$. The goal is to find an integral solution vector x such that $Ax = b$ while minimizing the objective function $w^\top x$. Sometimes, lower and upper bounds $\ell \in \mathbb{Z}^{d_2}$ and $u \in \mathbb{Z}^{d_2}$ have to be satisfied, i. e., $\ell_i \leq x_i \leq u_i$ for each component $i \in \{1, \ldots, d_2\}$. Formally, the Integer Linear Programming problem is defined as $\min\{w^\top x \mid Ax = b, \ell \leq x \leq u, x \in \mathbb{Z}^{d_2}\}$. We typically

write Δ for the largest absolute value of the entries in A, i. e., $\Delta = \|A\|_\infty$. It is known that these general ILPs are solvable in FPT if parameterized by the number of rows or columns in the constraint matrix by applying the well-known algorithms of Papadimitriou [143] or Lenstra and Kannan [106, 108].

As many applications are large in both dimensions, it is of great interest to find structures which are solvable efficiently also for this case. Here, we consider constraint matrices where the non-zero entries only appear inside blocks and these blocks are at certain positions in the matrix. We consider so-called 4-block n-fold ILPs. Here, the block matrices $C \in \mathbb{Z}^{r\times q}, A_1, \ldots, A_n \in \mathbb{Z}^{r\times t}$, $B_1, \ldots, B_n \in \mathbb{Z}^{s\times t}$ and $D_1, \ldots, D_n \in \mathbb{Z}^{s\times q}$ are arranged in the first rows, the first columns and along the diagonal of the constraint matrix, yielding:

$$\begin{pmatrix} C & A_1 & A_2 & \ldots & A_n \\ D_1 & B_1 & 0 & \ldots & 0 \\ D_2 & 0 & B_2 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ D_n & 0 & \ldots & 0 & B_n \end{pmatrix}.$$

The complete matrix has dimensions $d_1 = (r+ns)$ and $d_2 = (q+nt)$. Hence, the remaining variables w, u, ℓ have dimension $q + nt$ and $b \in \mathbb{Z}^{r+sn}$. It is still unknown whether these 4-block n-fold ILPs, i. e., the corresponding Integer Linear Programming problems, are solvable in FPT parameterized over q, r, s, t, Δ.

Two special cases considered in this work arise from the 4-block n-fold ILPs. If the matrices C and A only contain zero entries, i. e., they can be considered deleted, the resulting integer linear program is called 2-stage stochastic ILP and its constraint matrix is of the form:

$$\begin{pmatrix} D_1 & B_1 & 0 & \ldots & 0 \\ D_2 & 0 & B_2 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ D_n & 0 & \ldots & 0 & B_n \end{pmatrix}.$$

Thus, we have $r = 0$ yielding an $sn \times (q + nt)$ constraint matrix, $w, u, \ell \in \mathbb{Z}^{q+nt}$ and $b \in \mathbb{Z}^{sn}$. In contrast to before, this problem is known to be in FPT for the same parameterization [48].

If, in turn, we set C and D to zero, we get an n-fold ILP where the constraint matrix looks as follows:

$$\begin{pmatrix} A_1 & A_2 & \ldots & A_n \\ B_1 & 0 & \ldots & 0 \\ 0 & B_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & B_n \end{pmatrix}.$$

Here, we have $q = 0$ yielding a matrix of dimension $(r + ns) \times nt$. Further, $w, u, \ell \in \mathbb{Z}^{nt}$ and $b \in \mathbb{Z}^{r+sn}$. This case is also FPT:

Proposition 1 ([39]). *The n-fold Integer Linear Programming problem can be solved in time* $2^{O(rs^2)}(rs\Delta)^{O(r^2s+s^2)}\,(nt)^{1+o(1)}$.

Note that the 2-stage stochastic constraint matrix and the n-fold one are the transpose of each other.

CONFIGURATION ILPS Another type of integer linear programs are so-called *configuration ILPs*. A configuration determines which objects can be placed together. The variables of the ILP then correspond to configurations. These configuration ILPs are a classic tool in the design of algorithms for packing and scheduling problems.

For example, consider the Scheduling problem from above. We can rewrite the processing time vector $(p_1, \ldots, p_N)$ to $(a_1, \ldots, a_d)$ where d is the number of different processing times from the input and a_j is the number of jobs with processing time j. A configuration $\kappa = (\kappa_1, \ldots, \kappa_d)$ is a multiplicity vector stating that κ_j jobs with processing time j are selected. The objective is to choose M configurations such that all jobs are covered, i. e., the number of each processing time over all configurations equals the number of jobs with this processing time. Call the desired makespan T (which is either known or guessed in the case that we can estimate some lower and upper bounds on the makespan). Denote by $\mathcal{K}$ the set of feasible configurations, i. e., whose accumulated processing times are at most T and where $\kappa_j \leq a_j$ for each j. Let x_κ be a variable for each configuration $\kappa \in \mathcal{K}$. The configuration ILP looks as follows:

$$\sum_{\kappa \in \mathcal{K}} x_\kappa = M \tag{2.1}$$

$$\sum_{\kappa \in \mathcal{K}} x_\kappa \kappa_j = a_j \qquad \forall j \in \{1, \ldots, d\} \tag{2.2}$$

The first constraint ensures that we choose exactly M configurations. The second constraint is satisfied when we cover all jobs. Solving this ILP directly yields a feasible schedule for the desired makespan T: Each machine gets one of the chosen configurations; the configurations are resolved into their processing times – which can be seen as placeholders – and greedily filled by the corresponding jobs.

PROXIMITY AND SENSITIVITY For a point $x \in \mathbb{R}^{d_2}$ and a set $Y \subseteq \mathbb{R}^{d_2}$, define $\operatorname{dist}(x, Y)$ as the minimal ℓ_∞-distance of x to any point in Y, i. e., $\operatorname{dist}(x, Y) = \min_{y \in Y}\{\|x - y\|_\infty\}$. Furthermore, for two sets $X, Y \subseteq \mathbb{R}^{d_2}$ define $\operatorname{dist}(X, Y) = \max_{x \in X}\{\operatorname{dist}(x, Y)\}$ as the maximum over all minimal distances between some point $x \in X$ to the set Y. Consider the general ILP defined as above. Let $\operatorname{subDet}(A)$ be the largest determinant of any $d_3 \times d_3$ submatrix of A. Denote by $\operatorname{Sol.int}(A, b, w) = \arg\min\{w^\top x \mid Ax = b, x \in Z_{\geq 0}^{d_2}\}$ the set of its optimal integral solutions. In turn, $\operatorname{Sol.frac}(A, b, w) =$

$\arg\min\{w^\top z \mid Az = b, z \in Q^{d_2}_{\geq 0}\}$ is the set of optimal fractional solutions, i. e., the integrality constraint $x \in \mathbb{Z}^{d_2}$ is relaxed to $z \in \mathbb{Q}^{d_2}$. Next, we consider two important measures for ILPs which proved themselves useful in the design of various algorithms.

The *sensitivity* of the ILP measures the distance between two optimal integral solutions if the right-hand side changes. Formally, we define it as $\text{sens}(A, b, b', w) = \text{dist}(\text{Sol.int}(A, b, w), \text{Sol.int}(A, b', w))$.

Proposition 2 (Theorem 5 in [35]). *If* $\text{Sol.int}(A, b, w)$ *and* $\text{Sol.int}(A, b', w)$ *are non-empty, we have* $\text{dist}(x, \text{Sol.int}(A, b', w)) \leq (\|b - b'\|_\infty + 2) \cdot d_2 \cdot \text{subDet}(A)$ *for each* $x \in \text{Sol.int}(A, b, w)$.

This implies that $\text{sens}(A, b, b', w) \leq (\|b - b'\|_\infty + 2) \cdot d_2 \cdot \text{subDet}(A)$. The *proximity* of the ILP denoted by $\text{prox}(A, b, w)$ is formally defined as the term $\text{dist}(\text{Sol.frac}(A, b, w), \text{Sol.int}(A, b, w))$, i. e., the maximal distance between any optimal fractional solution and an optimal integral one.

Proposition 3 (Theorem 1 in [35]). *If* $\text{Sol.int}(A, b, w)$ *is non-empty, then we have* $\text{dist}(x, \text{Sol.frac}(A, b, w)) \leq d_2 \cdot \text{subDet}(A)$ *for each* $x \in \text{Sol.int}(A, b, w)$ *and further, we have* $\text{dist}(z, \text{Sol.int}(A, b, w)) \leq d_2 \cdot \text{subDet}(A)$ *for each* $z \in \text{Sol.frac}(A, b, w)$.

Note that this implies that $\text{prox}(A, b, w) \leq d_2 \cdot \text{subDet}(A)$. Further, by the Hadamard inequality, we get that $\text{subDet}(A) \leq \Delta^{d_3} \cdot d_3^{d_3/2}$ for any $d_3 \times d_3$ submatrix of A [79].

APPROXIMATION ALGORITHMS AND APPROXIMATION SCHEMES Finding an optimal solution can be costly. In such cases, we may be satisfied to compute a solution close to an optimal one. Denote by $\text{OPT}(I)$ the value of an optimal solution for some instance I. Further, let $A(I)$ be the value of a solution produced by an approximation algorithm A. We call A an ω-*approximation algorithm* if $\omega \cdot \text{OPT}(I) \geq A(I)$ holds for each instance I in the case of a minimization problem or $\text{OPT}(I) \leq \omega \cdot A(I)$ for maximization problems. We call ω the approximation ratio or ratio of the approximation algorithm A. Throughout this work, we assume that an approximation algorithm runs in polynomial time, i. e., we just consider *polynomial time approximation algorithms* but drop the name prefix.

A specific family of approximation algorithms are *approximation schemes*. They compute upon an input $\epsilon \in (0, 1]$ a $(1 + \epsilon)$-approximation, i. e., it is guaranteed that we find a solution $(1 + \epsilon)\text{OPT}(I) \geq A(I)$ for a minimization problem or $\text{OPT}(I) \leq (1 + \epsilon)A(I)$ otherwise. A *polynomial time approximation scheme* (PTAS) computes a solution in time $O(|I|^{f(1/\epsilon)})$ where f denotes an arbitrary function and $|I|$ is the encoding length of instance I. A PTAS is called an *efficient polynomial time approximation scheme* (EPTAS) if the running time is bounded by $f(1/\epsilon)|I|^{O(1)}$. An EPTAS is called *fully polynomial time approximation scheme* (FPTAS) if the function f is a polynomial. Note that EPTASs and FPTASs run in fixed-parameter tractable time for the parameter $1/\epsilon$ and thus, the underlying problem is FPT with parameter $1/\epsilon$.

The toolbox for designing approximation algorithms and schemes is rich. We want to point out two essential ideas which appear repeatedly throughout this work: dual approximation and preprocessing. In the *dual approximation framework* by Hochbaum and Shmoys [89], an optimization problem is solved indirectly. Instead of optimizing over a function, a procedure is developed which for a guess T on the optimal value either correctly states that there is no solution obtaining this value or computes a solution with value of at most $(1+\epsilon)T$. Having bounds on the feasible values of T, we can thus find the optimal one via a binary search and use the procedure in each iteration. Further, by *preprocessing* the instance, we simplify it. Commonly, we use *rounding* which introduces a small error, but reduces the types of different objects in the instance. For example, we round the processing times in the SCHEDULING problem such that we only have $f(1/\epsilon)$ many different ones. *Scaling* them further gives the nice property that these values are integral. Overall, these techniques allow us for example to model the instances as a configuration ILP with small parameters making them solvable efficiently.

Part I

INTEGER LINEAR PROGRAMS

It's the questions we can't answer that teach us the most. They teach us how to think. If you give a man an answer, all he gains is a little fact. But give him a question and he'll look for his own answers.

— from *A wise man's fear* by Patrick Rothfuss

3

NEAR-LINEAR TIME ALGORITHM FOR n-FOLD ILPS

This chapter considers n-fold ILPs. Recall that this class of integer linear programs has a constraint matrix $\mathcal{A}$ with a specific block structure where non-zero entries only appear in the first r rows in blocks of size $r \times t$ and in blocks of size $s \times t$ along the diagonal underneath, yielding:

$$\mathcal{A} = \begin{pmatrix} A_1 & A_2 & \dots & A_n \\ B_1 & 0 & \dots & 0 \\ 0 & B_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & B_n \end{pmatrix}.$$

We subdivide a solution x into parts of length t, which naturally correspond to the block structure. We call them *bricks* and denote by $x^{(i)}$ the ith one. The corresponding sub-matrices $(A_i B_i)^\top$ in $\mathcal{A}$ are called *blocks*. Throughout this chapter, we assume a lower bound $\Delta \geq 2$ on the largest absolute value of the entries of $\mathcal{A}$. The assumption is only used because ignoring the corner-case $\Delta = 1$ allows cleaner bounds on the running time.

Lately, n-fold ILPs have received great attention [8, 39, 47, 97, 115, 119] and were studied intensively for two reasons. First, many optimization problems are expressible as n-fold ILPs [83, 97, 100, 115, 126]. Secondly, n-fold ILPs can be solved much more efficiently than arbitrary ILPs [39, 47, 83, 119].

The main idea for the previous algorithms relies on some insight about the Graver bases of n-fold ILPs which are special elements of the kernel of $\mathcal{A}$. More formally, we introduce the following definitions:

Definition 1. *The* kernel *of a matrix $A \in \mathbb{Z}^{d_1 \times d_2}$ is defined as the set of integral vectors $x \in \mathbb{Z}^{d_2}$ with $Ax = 0$. We write $\ker(A)$ for them.*

Definition 2. *An element $g \in \ker(A)$ is a Graver (basis) element if it is not the sum of two sign-compatible, non-zero elements $u, v \in \ker(A)$.*

Here, sign-compatible means that $u_i \cdot v_i \geq 0$ for every i.

Proposition 4 ([34])**.** *Let $A \in \mathbb{Z}^{d_1 \times d_2}$ and let $x \in \ker(A)$. Then there exist $2d_2 - 1$ Graver basis elements $g_1, \dots, g_{2d_2-1}$ which are sign-compatible with x such that*

$$x = \sum_{i=1}^{2d_2-1} \lambda_i g_i$$

for some $\lambda_1, \dots, \lambda_{2d_2-1} \in \mathbb{Z}_{\geq 0}$.

Many results for n-fold ILPs rely on the fact that the ℓ_1-norm of Graver basis elements for n-fold matrices are small. The best bound known for the ℓ_1-norm is due to Eisenbrand et al. [47].

Proposition 5 ([47]). *The ℓ_1-norm of the Graver basis elements of an n-fold matrix $\mathcal{A}$ is bounded by $O(rs\Delta)^{rs}$.*

The best algorithm previous to ours has a running time of $(rs\Delta)^{O(r^2s+rs^2)} L(nt)^2 \log^2(nt) + \mathrm{LP}$ and is due to Eisenbrand et al. [47]. Here, L denotes the encoding length of the largest number in the instance and LP denotes the time required for solving the corresponding LP relaxation. This augmentation algorithm is the last one in a line of research where local improvement/ augmenting steps are used to converge to an optimal solution. Their key result is an improvement of the bound on the Graver basis elements.

Nevertheless, the dependence on n in the algorithm above is still large. Indeed, in practice, a quadratic running time is simply not suitable for large data sets [9, 111, 162]. For example, when analyzing big data, large real world graphs as in telecommunication networks or DNA strings in biology, the duration of the computation would go far beyond the scope of an acceptable running time [9, 111, 162]. For this reason, even problems which can be solved in quadratic running time are still studied from the viewpoint of approximation algorithms with the objective of obtaining results in sub-quadratic time, even at the cost of a worse quality [9, 111, 162]. Hence, it is an intriguing question whether the quadratic dependency on nt, i. e., on the number of variables, can be eliminated. We are the first to answer this question affirmatively. The technical novelty comes from a surprising area: We use a combinatorial structure called splitters which has been used to derandomize Color Coding algorithms [7]. It allows us to build a powerful data structure that is maintained during the local search and from which we can derive an improving direction in logarithmic time.

These splitters (see e.g. [136]) are commonly used in the FPT community, but they have not been applied in the context of n-fold ILPs so far. Splitters are closely related to hash functions. Hash functions map elements from a potentially huge set to a much smaller set. They can usually be computed deterministically, but mimic the behavior of a random mapping.

Definition 3. *An (d, k_1, k_2) splitter is a family of hash functions F from $\{1,\dots,d\}$ to $\{1,\dots,k_2\}$ such that for every $S \subseteq \{1,\dots,d\}$ with $|S| = k_1$, there exists a function $f \in F$ that splits S evenly, that is, for every $j, j' \leq k_2$ we have $|f^{-1}(j) \cap S|$ and $|f^{-1}(j') \cap S|$ differ by at most 1.*

If $k_2 \geq k_1$, the above means that there is some hash function that has no collisions when restricted to S. A hash function from $\{1,\dots,d\}$ to $\{1,\dots,k_2\}$ naturally corresponds to a partition of the set $\{1,\dots,d\}$ into exactly k_2 subsets. In other words, the definition states that there is one such partition where each subset contains almost the same number of elements of S. We refer to $|F|$, i. e., the number of hash functions in a splitter, as the size of the splitter. Remarkably, splitters of very small size exist.

Proposition 6 ([7, 136]). *There exists an (d, k, k^2) splitter of size $k^6 \log(k) \log(d)$ which is computable in time $k^6 \log(k) d \log(d)$.*

An alternative approach to the result above is to use FKS hashing, a particular hash function defined in the proposition below. Although it requires an extra factor of $\log(d)$ compared to the result above, it is particularly easy to implement.

Proposition 7 (Corollary 2 and Lemma 2, [62]). *Define for every prime number $q_1 < k^2 \log(d)$ and prime number $q_2 < q_1$ the hash function $x \mapsto 1 + (q_2 \cdot (x \bmod q_1) \mod k^2)$. This is an (d, k, k^2) splitter of size $O(k^4 \log^2(d))$.*

Having these at hand, we briefly elaborate the main technical novelty. Let x be some feasible, non-optimal solution for the n-fold ILP. It is clear that when y^* is an optimal solution for $\max\{w^\top y \mid \mathcal{A}y = 0, \ell - x \leq y \leq u - x, y \in \mathbb{Z}^{nt}\}$ then $x + y^*$ is optimal for the initial n-fold ILP. In other words, y^* is a particularly good improving step. A natural approximation of y^* is to consider directions y of small size and multiply them by some step length, i. e., to find some λy with $\|y\|_1 \leq k$ for a value k depending only on Δ, r, and s. Due to its norm, at most k components of y are non-zero. This implies that at most k of the n blocks are used in y. If we randomly color the blocks with k^2 colors, then with high probability at most one block of every color is used. This reduces the problem to choosing a solution of a single brick for every color and to aggregate them. We add data structures for every color to implement this efficiently. Of course, there is a chance that the colors do not split y perfectly. We handle this by using a deterministic structure of multiple colorings (instead of one) so that it is guaranteed that at least one of them has the desired property.

To design the final algorithm, we still have to handle unbounded variables in an n-fold ILP. Indeed, this is a non-trivial issue in the previous algorithms from literature such as those mentioned above. Usually, the LP relaxation is solved and proximity results are used to introduce artificial bounds. At the time this work was first published, is was not known whether n-fold LPs can be solved in near-linear time with respect to the number of variables. Hence, it was an obstacle for obtaining a near-linear running time. Further, even the current state-of-the-art algorithm for solving the relaxation might dominate the running time for solving the n-fold ILP with finite variable bounds [39]. We avoid solving the LP by bounding the variables in a function of the given finite upper bounds and the right-hand side of the n-fold ILP.

SUMMARY OF RESULTS

- We present an algorithm which solves n-fold ILPs in time $(rs\Delta)^{O(r^2s+s^2)} Lnt \log^5(nt) + \text{LP}$ where LP is the time to solve the LP relaxation of the n-fold ILP. A notable aspect is the near-linear dependence on the number of variables. The crucial step is to speed up the computation of the improving directions.
- We avoid solving the LP relaxation. This leads to a purely combinatorial algorithm with running time $(rs\Delta)^{O(r^2s+s^2)} L^2 nt \log^7(nt)$.

- The running time with respect to the dependence on the parameters, i. e., $(rs\Delta)^{O(r^2s+s^2)}$, improves on the function $(rs\Delta)^{O(r^2s+rs^2)}$ in the previous best algorithms.

FURTHER RELATED WORK The first XP-time algorithm for solving n-fold integer programs is due to De Loera et al. [126] with a running time of $n^{g(\mathcal{A})}L$, i. e., it admits an algorithm that runs in time polynomial in the encoding length of the input where the exponent is dependent on the parameters. Here, $g(\mathcal{A})$ denotes the so-called Graver complexity of the constraint matrix $\mathcal{A}$ and L is again the encoding length of the largest number in the input. This algorithm already uses the idea of iteratively converging to an optimal solution by finding improving directions. Nevertheless, the Graver complexity can be huge even for small n-fold integer linear programs and thus this algorithm was of no practical use [83]. The exponent of this algorithm was improved to a constant by Hemmecke et al. [83] yielding the first cubic time algorithm for solving n-fold ILPs. More precisely, the running time of their algorithm is $\Delta^{O(t(rs+st))}L(nt)^3$, i. e., fixed-parameter tractable when parameterized over $\Delta, r, s,$ and t. Lately, two more improvements were obtained. One of the results is due to Koutecký et al. [119], who gave a strongly polynomial algorithm with running time $\Delta^{O(r^2s+rs^2)}(nt)^6\log(nt) + \mathrm{LP}$. Recall that LP is the running time for solving the corresponding LP relaxation, which is strongly polynomial since the entries of the matrix are bounded. Eisenbrand et al. [47] simultaneously reduced the dependency on nt from cubic to quadratic by introducing new proximity results and stronger bounds on the Graver norm. This leads to an algorithm with running time $(\Delta rs)^{O(r^2s+rs^2)}L(nt)^2\log^2(nt) + \mathrm{LP}$. Both results require only a polynomial dependency on t. Concurrently to our work, Eisenbrand et al. improved the previous results and obtained an algorithm with running time $(\Delta rs)^{O(r^2s+rs^2)}Lnt\log(nt)$ [48]. Note that their dependency on the poly-logarithmic term is smaller than ours, however, in turn, the dependency on the parameters in the exponent is worse compared to our work. Recently, a new algorithm was developed by Cslovjecsek et al. [39]. This is the first algorithm to run in strongly polynomial and near-linear time. In contrast to previous works, their algorithm does not rely on the augmentation framework. Instead, they use a stronger ILP relaxation which yields a better proximity. Exploiting the LP techniques by Norton et al. [139] they managed to solve the stronger relaxation fast and search for the optimal ILP solution efficiently by considering the small box implied by the proximity.

For applications of n-fold ILPs in string problems, flow problems, or scheduling problems, we refer the reader to the publications [83, 84, 97, 101, 117, 126, 140] and the references within.

STRUCTURE OF THIS CHAPTER Section 3.1 gives the algorithm for efficiently computing the augmenting steps. This is then integrated into an algorithm for n-fold ILPs in Section 3.2. Initially, we assume finite variable bounds are given and then discuss how to eliminate this assumption using the solution of the LP relaxation. Finally, in Section 3.3, we discuss how to handle

infinite variable bounds without the LP relaxation and give new structural results.

3.1 EFFICIENT COMPUTATION OF IMPROVING DIRECTIONS

The backbone of our algorithm is the fast computation of augmenting steps. It relies on the observation that we can update the augmenting steps very efficiently if the input changes only slightly. In other words, whenever we change the current solution by applying an augmenting step, we do not have to recompute the next augmenting step from scratch. The augmenting steps depend on a partition of the blocks. In the following, we define the notion of a best step based on a fixed partition. Later, we find steps for a number of partitions independently and take the best among them.

Definition 4. *Let $P = \{P_1, \dots, P_p\}$ be a partition of the n blocks. Let $\overline{u} \in \mathbb{Z}^{nt}_{\geq 0}$ and $\overline{\ell} \in \mathbb{Z}^{nt}_{\leq 0}$ be some upper and lower bounds on the variables (not necessarily the same as in the n-fold ILP). Denote by $\mathcal{P}_j$ the set of all indices in P_j (i. e., the indices of all columns of all blocks in P_j). Further, let $\mathcal{S}^j_i$ be the set of all indices of the columns in block $i \in P_j$. A (P,k)-best step is an optimal solution $x \in \mathbb{Z}^{nt}$ (and implicitly a solution for $g_j, j \in \{1,\dots,p\}$) of the system below.*

$$
\begin{aligned}
\max\ & w^\top x && \\
& \mathcal{A}x = 0 && \\
& \sum_{h \in \mathcal{S}^j_{g_j}} |x_h| \leq k && \forall j \in \{1,\dots,p\} \\
& x_h = 0 && \forall j \in \{1,\dots,p\}, h \in P_j \setminus \{\mathcal{S}^j_{g_j}\} \\
& \overline{\ell} \leq x \leq \overline{u}. &&
\end{aligned}
$$

This means a (P,k)-best step is an element of kernel($\mathcal{A}$) which uses only one block g_j of every $P_j \in P$. Within that block, the norm of the solution must be at most k. Later, we choose k as the upper bound for the ℓ_1-norm of a Graver basis element, i. e., $k = O(rs\Delta)^{rs}$.

Theorem 8. *Let P be a partition of the n blocks into k^2 sets. Consider the problem of finding a (P,k)-best step in an n-fold matrix where the lower and upper bounds $\overline{\ell}, \overline{u}$ can change. This problem can be solved initially in time $k^{O(r)}\Delta^{O(r^2+s^2)}nt$ and then in $k^{O(r)}\Delta^{O(r^2+s^2)}\log(nt)$ update time whenever the bounds of a single variable change.*

Proof. Let P be a partition of the bricks from matrix $\mathcal{A}$ into k^2 disjoint sets $P_1, P_2, \dots, P_{k^2}$. Solving the (P,k)-best step problem requires that we choose from each set $P_j \in P$ at most one block and set this brick's variables. All variables in other bricks of P_j must be 0.

Let x be a (P,k)-best step and let $x^{(j)}$ have the values of x in the variables of P_j and 0 in all other variables. Then, by definition, $\|x^{(j)}\|_1 \leq k$ holds. This implies that the right-hand side regarding $x^{(j)}$, that is to say, $\mathcal{A}x^{(j)}$, is also

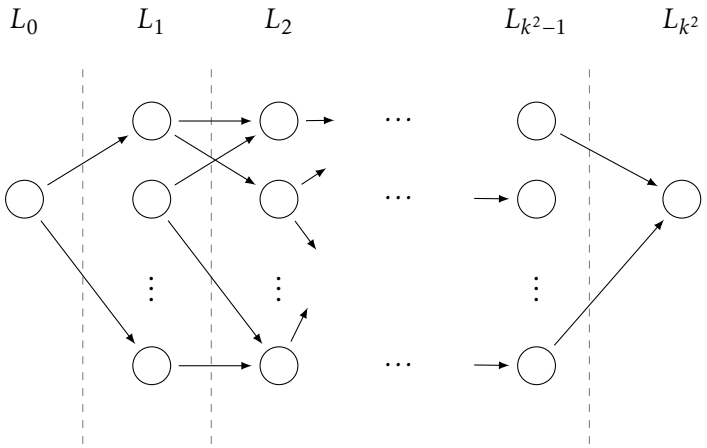

Figure 3.1: This figure shows an example for a layered graph obtained while solving the (P,k)-best step problem. There are k^2+1 layers visually separated by gray dashed lines. This includes one source layer L_0 and one target layer L_{k^2}, both with just a single node representing the zero sum. Further, there are k^2-1 layers with $(2k^3\Delta+1)^r$ nodes each, where in one layer the nodes stand for all reachable partial sums. Two nodes v_1, v_2 from adjacent layers L_{j-1}, L_j are connected if the difference of the corresponding partial sums, namely $v_2 - v_1$, can be obtained by a solution y of variables from only one block of P_j (with $\|y\|_1 \leq k$). The weight of the edge is the largest gain for the objective function $w^\top y$ over all possible bricks.

small. Since the absolute value of an entry in $\mathcal{A}$ is at most Δ, we have that $\|\mathcal{A}x^{(j)}\|_\infty \leq k\Delta$. Let a_i be the ith row of $\mathcal{A}$. If $i > r$, then $a_i x^{(j)} = 0$. This is because $\mathcal{A}x = 0$ and a_i has all its support either completely inside P_j or completely outside P_j. Thus, the value of $\mathcal{A}x^{(j)}$ is one of the $(2k\Delta+1)^r$ many values we get by enumerating all possibilities for the first r rows. Furthermore, since P has only k^2 sets, the partial sum $\mathcal{A}(x^{(1)}+\cdots+x^{(j)})$ is always one of $(2k^3\Delta+1)^r = (k\Delta)^{O(r)}$ many candidates.

Hence, to find a (P,k)-best step, we can restrict our search to solutions whose partial sums stay in this range. To do so, we set up a graph containing k^2+1 layers $L_0, L_1, \ldots L_{k^2-1}, L_{k^2}$. An example is given in Figure 3.1. The first layer L_0 consists of just one node marking the start with partial sum zero. Similarly, the last layer L_{k^2} just contains the target node also having partial sum zero since a (P,k)-best step is an element of kernel$(\mathcal{A})$. Each layer L_j with $1 \leq j \leq k^2-1$ contains $(2k^3\Delta+1)^r$ many nodes, each representing one possible value of $\mathcal{A}(x^{(1)}+\cdots+x^{(j)})$. Two nodes v_1, v_2 from adjacent layers L_{j-1}, L_j for all $1 \leq j \leq k^2$ are connected if the difference of the corresponding partial sums, namely $v_2 - v_1$, can be obtained by a solution y of variables from only one brick of P_j (with $\|y\|_1 \leq k$). The weight of the edge is the largest gain for the objective function $w^\top y$ over all possible bricks. Hence, it could be necessary to compute and compare up to n values for each P_j and each difference in the partial sums to insert one edge into the graph.

Finally, we just have to find the longest path in this graph, as it corresponds to a (P,k)-best step. The out-degree of each node is bounded by $(2k^3\Delta+1)^r$

since at most this many nodes are reachable in the next layer. Therefore, the overall number of edges is bounded by

$$(k^2+1)(2k^3\Delta+1)^r(2k^3\Delta+1)^r = (k\Delta)^{O(r)}.$$

Using the Bellman-Ford algorithm, we can solve the Longest Path problem for a graph with $|V|$ vertices and $|E|$ edges in time $O(|V|\cdot|E|)$, as the graph is directed and acyclic. This gives a running time of $(k\Delta)^{O(r)}\cdot(k\Delta)^{O(r)} = (k\Delta)^{O(r)}$ for solving the problem. Constructing the graph, however, requires solving integer linear programs of the form

$$\begin{aligned} &\max w'^{\top}x \\ &\begin{pmatrix} A_j \\ B_j \end{pmatrix} x = \begin{pmatrix} b' \\ 0 \end{pmatrix} \\ &\|x\|_1 \le k \\ &\ell' \le x \le u' \\ &x \in \mathbb{Z}^t \end{aligned}$$

for each pair of vertices v_1, v_2 in consecutive layers. Here $b' \in \mathbb{Z}^r$ is the difference of the partial sums associated with v_1 and v_2 and ℓ', u', w' are the upper and lower bounds and the cost vector of the block.

Claim 9. *After initialization in time $kt\Delta^{O(r^2+s^2)}$, the system above can be solved in time $k\log(t)\Delta^{O(r^2+s^2)}$. Moreover, if for one variable the bounds ℓ'_i and u'_i change, we can update the solution in the same time.*

This can be achieved by carefully applying the algorithm by Eisenbrand and Weismantel [49]. We postpone this to after the main proof. The number of integer linear programs to solve is at most n times the number of edges since we have to compare the values of up to n blocks. This gives a running time of

$$O(nkt) + n(k\Delta)^{O(r)}\log(t)\Delta^{O(r^2+s^2)} \le nt\cdot k^{O(r)}\Delta^{O(r^2+s^2)}$$

for constructing the graph.

To obtain the update time from the statement of the theorem, we can solve the Longest Path problem again, but we cannot construct the graph from scratch. However, in order to construct the graph, we still have to find the best value over all blocks for each edge. Fortunately, if only a few blocks are updated (in their lower and upper bounds), it is not necessary to recompute all values. Each edge corresponds to a particular $P_j \in P$ and a fixed right-hand side (a possible value of $\mathcal{A}x^{(j)}$). We require an appropriate data structure $\mathcal{D}_e$ for every edge e which supports fast computation of the operations FindMax, Insert, and Delete. An AVL tree computes each of these operations in time $O(\log(\mathcal{N}))$ where $\mathcal{N}$ is the number of elements [36]. For an introduction to AVL trees and the operations, we recommend [36].

We store pairs (v,i) in $\mathcal{D}_e$ where i is a block in P_j and v is the maximum increase in the objective value using block i with respect to a right-hand side

corresponding to e. The pairs are stored in lexicographical order. Since there are at most n blocks in P_j, the data structure has at most n elements. Initially, we can build $\mathcal{D}_e$ in time $nt\Delta^{O(r^2+s^2)}$ for each edge e.

Now consider a change to the instance. Recall that we are looking at changes that affect only a single block, namely the upper and lower bounds within that block change. We update the data structure $\mathcal{D}_e$ for each edge e to reflect the changes and we recompute the edge value of each edge e using $\mathcal{D}_e$. Then, we simply solve the Longest Path problem again. Let $P_j \in P$ be the set that contains the block i that has changed in some variable. We only have to consider edges from L_{j-1} to L_j since none of the other edges are affected by the change. For such a relevant edge e, we replace the current value (v,i) by the value (v',i) that the brick i would produce considering the changed bounds. In detail, we have to find the entry (v,i) in $\mathcal{D}_e$ which can be done in time $O(\log(n))$ in AVL trees with n entries. Then, we remove (v,i) from $\mathcal{D}_e$ in time $O(\log(n))$ and finally, insert (v',i) in the same time. The running time to update $\mathcal{D}_e$ for one edge thus is

$$\underbrace{k\log(t)\Delta^{O(r^2+s^2)}}_{\text{Calculate } v \text{ for changed block}} + \underbrace{O(\log(n))}_{\text{FIND, INSERT and DELETE tupels}} \leq k\log(nt)\Delta^{O(r^2+s^2)}.$$

In order to update the edge value of e using $\mathcal{D}_e$, we simply have to find the maximum element in $\mathcal{D}_e$. The operation FINDMAX can be computed in time $O(\log(n))$. To summarize, the total time to update the (P,k)-best step after the bounds of a single block changed consists of (1) updating each $\mathcal{D}_e$, (2) finding the maximum in each $\mathcal{D}_e$ and (3) solving the Longest Path problem. We conclude that the update time is

$$\underbrace{(k\log(nt)\Delta^{O(r^2+s^2)}}_{\text{Updating } \mathcal{D}_e} + \underbrace{\log(n))}_{\text{FINDMAX in } \mathcal{D}_e} \underbrace{(k\Delta)^{O(r)}}_{\text{Number of edges}} + \underbrace{(k\Delta)^{O(r)}}_{\text{Longest Path problem}}$$
$$\leq k^{O(r)}\Delta^{O(r^2+s^2)}\log(nt). \quad \square$$

Proof of Claim 9. We first linearize the constraint on the ℓ_1-norm of x by writing each variable x_i as the difference of two non-negative variables $x_i^+ - x_i^-$. Thus we obtain the following equivalent integer linear program.

$$\begin{aligned}
&\max\ w'^T(x^+ - x^-)\\
&\begin{pmatrix} A_j \\ B_j \end{pmatrix}(x^+ - x^-) = \begin{pmatrix} b' \\ 0 \end{pmatrix}\\
&\sum_{i=1}^{t}[x_i^+ + x_i^-] \leq k\\
&\max\{0,\ell_i'\} \leq x_i^+ \leq \max\{0,u_i'\}\\
&\max\{0,-u_i'\} \leq x_i^- \leq \max\{0,-\ell_i'\}\\
&x \in \mathbb{Z}^t_{\geq 0}
\end{aligned}$$

This system has $2t$ bounded variables, $r+s+1$ additional constraints, and entries of the matrix bounded by Δ in absolute value. Using the algorithm by Eisenbrand and Weismantel [49] solving one of them requires time

$$t \cdot O(r+s+1)^{r+s+4} \cdot O(\Delta)^{(r+s+1)(r+s+4)} \log^2((r+s+1)\Delta) + \mathrm{LP} \\ = t\Delta^{O(r^2+s^2)} + \mathrm{LP}.$$

where LP is the time for solving the LP relaxation. The simplification in the running time comes from the calculation

$$O(r+s+1)^{O(r+s)} \leq 2^{O((r+s)\log(r+s+1))} \leq 2^{O((r+s)^2)} \leq 2^{O(r^2+s^2)}$$

and that — by the assumption of $\Delta \geq 2$ — this term can be absorbed by $\Delta^{O(r^2+s^2)}$.

Furthermore, we can reduce the dependency on t: Since the number of constraints in the ILP above is very small, there are only $\Delta^{O(r+s)}$ many different columns. Due to the cardinality constraint $\sum_{i=1}^{t}[x_i^+ + x_i^-] \leq k$, we only have to consider $2k$ many variables of each type of column, namely the k many x_i^+ with $u_i' > 0$ and maximal w_i' and the k many x_i^- with $\ell_i' < 0$ and minimal w_i'. If some solution uses a variable not in this set, then by the pigeonhole principle — that is, if a elements are mapped to $b < a$ values, then at least two elements are mapped to the same value — there is a variable with the same column values and a superior objective value which can be increased or decreased. We can reduce the variable outside this set and increase the corresponding variable inside this set until all variables outside the set are zero. Hence, after filtering out all dominated variables, we can substitute t for $2k\Delta^{O(r^2+s^2)}$ in the running time above. In order to handle changes in the variable bounds, we have to rely on an additional data structure.

We use AVL trees (or similar data structures) to maintain a set of all variables with $u_i' > 0$ ($\ell_i' < 0$) such that we can find the k best among them in time $O(k\log(t))$. Whenever the bounds of some variable change, that is, u_i' or ℓ_i' changes from zero to non-zero or vice versa, we might have to add or remove entries which also takes only logarithmic time. After initialization in time $kt\Delta^{O(r^2+s^2)}$ solving such an integer linear program (or updating it) can therefore be implemented in time

$$k\log(t) + 2k\Delta^{O(r+s)}\Delta^{O(r^2+s^2)} + \mathrm{LP} \leq k\log(t)\Delta^{O(r^2+s^2)} + \mathrm{LP} \\ \leq k\log(t)\Delta^{O(r^2+s^2)}.$$

The last inequality holds because, using Tardos's algorithm [159], LPs can be solved in time polynomial in the encoding size of the matrix. The encoding size of the matrix can be bounded by $2k\Delta^{O(r+s)}(r+s)\log(\Delta)$. This is dominated by the other term. □

3.2 THE AUGMENTING STEP ALGORITHM

This section presents the complete algorithm for the case where all lower and upper bounds are finite. Later, we explain the necessary steps to deal with

infinite bounds. But first, we show the idea of converging from a given initial feasible solution to an optimal one. To compute the initial solution, we apply the same algorithm on a slightly modified instance. This approach resembles the procedure in the literature (see for example [47, 83, 119]), although we apply the results from the previous section to speed up the computation of augmenting steps.

Let x be a feasible solution for the n-fold ILP, in particular, $\mathcal{A}x = b$. Let x^* be an optimal one. Proposition 4 states that we can decompose the difference vector $x' = x^* - x$ into at most $2nt - 1$ weighted Graver basis elements, i. e.,

$$x' = x^* - x = \sum_{j=1}^{2nt-1} \lambda_j g_j.$$

For intuition, consider the following simple approach (this is similar to the algorithm by Hemmecke et al. [83]). Suppose we are able to guess an optimal vector $\lambda_i g_i = \operatorname{argmax}_j\{w^\top(\lambda_j g_j)\}$ regarding the gain for the objective function. This pair of step length λ_i and Graver element g_i is called the *optimal Graver step*. Then, we can augment the current solution x by adding $\lambda_i g_i$ to it, i. e., we set $x \leftarrow x + \lambda_i g_i$. Feasibility follows because all g_i are sign-compatible. This procedure is repeated until no improving step is possible and therefore, x must be optimal. In each iteration, this decreases the gap to the optimal solution by a factor of at least $1 - 1/(2nt)$ by the pigeonhole principle. It may be costly to guess an optimal Graver step, but for our purposes it suffices to find an augmenting step that is approximately as good, i. e., whose gain is only smaller by a constant factor with respect to an optimal one.

However, we have to guess λ_i first. Since $x + \lambda_i g_i$ is feasible, we have that $\lambda_i g_i \le u - x \le u - \ell$ and $\lambda_i g_i \ge \ell - x \ge \ell - u$. Let $j \in \operatorname{supp}(g_i)$ be an index of some non-zero variable, i. e., j is in the *support* of g_i. If $(g_i)_j > 0$ holds, then $\lambda_i \le (\lambda_i g_i)_j \le u_j - \ell_j$. Otherwise $(g_i)_j < 0$ and $\lambda_i \le -(\lambda_i g_i)_j \le -(\ell_j - u_j) = u_j - \ell_j$. Hence, it suffices to try all values in the range $\{1, \dots, \Gamma\}$ where $\Gamma = \max_j\{u_j - \ell_j\}$. Proceeding as in [47], we improve the running time further by not taking every value into consideration. Instead, we look at guesses of the form $\lambda' = 2^k$ for $k \in \{0, \dots, \lfloor \log(\Gamma) \rfloor\}$. Doing so, we lose a factor of at most 2 regarding the improvement of the objective function as $w^\top(\lambda' g_i) > 1/2 \cdot w^\top(\lambda_i g_i)$ when taking $\lambda' = 2^{\lfloor \log(\lambda_i) \rfloor} > \lambda_i/2$. Fix λ' to the value above. Next, we compute an augmenting step that is at least as good as $\lambda' g_i$. It holds that g_i is a solution of the inequality system

$$\begin{aligned} \mathcal{A}y &= 0 \\ \|y\|_1 &\le k \\ \lceil \frac{\ell - x}{\lambda'} \rceil \le y &\le \lfloor \frac{u - x}{\lambda'} \rfloor \end{aligned}$$

where $k = O(rs\Delta)^{rs}$ is the bound on the norm of Graver elements from Proposition 5. Suppose we have guessed some partition $P = \{P_1, \dots, P_{k^2}\}$ of the blocks such that of each P_j only a single brick has non-zero variables in g_i. Clearly, the augmenting step $\lambda' y^*$ where y^* is a (P,k)-best step with bounds $\overline{\ell} = \lceil \frac{\ell - x}{\lambda'} \rceil$ and $\overline{u} = \lfloor \frac{u - x}{\lambda'} \rfloor$ would be at least as good as $\lambda' g_i$. Indeed,

Theorem 8 explains how to compute such a (P,k)-best step dynamically and when we add $\lambda' y^*$ to x, we only change the bounds of at most k^3 many variables. Hence, it is very efficient to recompute (P,k)-best steps until we have converged to an optimal solution. However, valid choices of λ' and P might be different in every iteration. Regarding λ', we simply compute (P,k)-best steps for each of the $O(\log(\Gamma))$ many guesses and take the best among them. We proceed similarly for P by guessing a small number of partitions and guarantee that always at least one of them is valid. For this purpose, we employ splitters. More precisely, we compute a (n,k,k^2) splitter of the n blocks. Since g_i has norm bounded by k, it can also only use at most k bricks. Therefore, the splitter always contains a partition $P = \{P_1,\dots,P_{k^2}\}$ where g_i only uses a single brick in every P_j.

Summarizing, in every iteration, we solve a (P,k)-best step problem for every guess λ' and every partition P in the splitter and take the overall best solution as an improving direction $\lambda' y^*$. Then, we update our solution x by adding $\lambda' y^*$ onto it. At most k^2 many bricks change (and within each brick only k variables can change) and therefore, we can efficiently recompute the (P,k)-best steps for every guess for the next iteration. This way, we guarantee that we improve the solution by a factor of at least $1 - 1/(4nt)$ in every iteration. The explicit running time of these steps is analyzed in the next theorem.

Recall that we still have to find an initial solution. This solution can be computed by using the augmenting step algorithm described above. To obtain an initial solution for our n-fold ILP, we construct a new n-fold ILP which has a trivial feasible solution and whose optimal solution corresponds to a feasible solution of the original problem. This approach resembles the first phases of the classical two-phase Simplex method and is also used for example in [47, 83, 119].

First, we extend our n-fold matrix $\mathcal{A}$ by adding $(r+s)n$ new columns as follows: After the first block $(A_1, B_1, 0, \dots, 0)^\top$, add $r+s$ columns. The first r ones contain an $r \times r$ identity matrix which we denote by $\mathcal{I}_r$. The next s columns contain an $s \times s$ identity matrix $\mathcal{I}_s$. This submatrix starts at row $r+1$. Again all other entries are zero in these columns. After the next block, we again introduce $r+s$ new columns, the first r ones containing just zeros, the next an $\mathcal{I}_s$ matrix at the height of B_2. We repeat this procedure of adding $r+s$ columns after each block. The first r only have zero entries and the next s containing $\mathcal{I}_s$ at the height of B_i. The resulting matrix $\mathcal{A}^{\text{init}}$ looks as follows:

$$\mathcal{A}^{\text{init}} = \begin{pmatrix} A_1 & \mathcal{I}_r & 0 & A_2 & 0 & 0 & \cdots & A_n & 0 & 0 \\ B_1 & 0 & \mathcal{I}_s & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & B_2 & 0 & \mathcal{I}_s & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & & B_n & 0 & \mathcal{I}_s \end{pmatrix}.$$

Due to our careful extension, $\mathcal{A}^{\text{init}}$ again has n-fold structure. For clarity, the relevant submatrices are framed in the matrix above. Note that zero entries inside of a block do no harm. It may seem that for the right-hand side b, we now have a trivial solution consisting only of the new columns. However, old variables have upper and lower bounds and thus setting them to 0 might not be possible. In order to handle this case, we subtract ℓ, the lower bound, from all upper and lower bounds and set the right-hand side to $b' = b - \mathcal{A}\ell$. We get an equivalent n-fold ILP where every solution is shifted by ℓ. Now, we can find a feasible solution y' (for b') using solely the new variables by defining y' as the transpose of

$$(0,\dots,0,b'_1,\dots b'_{r+s},0,\dots,0,b'_{r+s+1},\dots,b'_{r+2s},0,\dots,0,b'_{r+ns-s+1},\dots,b'_{r+ns})$$

where each non-zero entry corresponds to the columns containing the submatrices $\mathcal{I}_r$ and $\mathcal{I}_s$ respectively with a multiplicity of the remaining right-hand side b'. Next, we introduce a cost vector that penalizes using the new columns by having non-zero entries w'_i corresponding to the positions of the new variables. We set

$$w^{\text{init}} = (0,\dots 0,w'_1,\dots w'_{r+s},0,\dots 0,w'_{r+ns-s+1},\dots,w'_{r+ns})^\top$$

where the zero entries correspond to old variables. The values w'_i and the lower and upper bounds for the new variables depend on the sign of the right-hand side.

- If $b'_i \geq 0$, set $w'_i = -1$, the lower bound to 0 and the upper bound to b'_i. This way the variable can only be non-negative.
- If $b'_i < 0$, set $w_i = 1$, the lower bound to b'_i and the upper bound to 0. This way the variable can only be non-positive.

Clearly, a solution has a value of 0 if and only if none of the new columns are used and no solution of better value is possible. If we use our augmenting step algorithm and solve this problem (i. e., find a solution vector that maximizes the objective value), we either find a solution yielding a value of 0 or one yielding a negative value. In the former case, we have obtained a solution using none of the new columns (their corresponding component in the solution is zero), as they would add a negative term to our objective function. Therefore, we can delete the new columns and obtain a solution for the original problem (after adding ℓ to it). If the algorithm does not find a solution with value 0, then there cannot be a feasible solution for the original problem. Hence, we have determined infeasibility.

Theorem 10. *The dynamic augmenting step algorithm described above computes an optimal solution for the n-fold* Integer Linear Programming *problem in time $(rs\Delta)^{O(r^2s+s^2)}O(L^2nt\log^5(nt))$ when finite variable bounds are given for each variable. Here L is the encoding length of the largest number occurring in the input.*

Proof. Due to Proposition 4, we know that the difference vector $x' = x^* - x$ of an optimal solution x^* to our current solution x can be decomposed into $2nt - 1$ weighted Graver basis elements. Hence, if we adjust our solution x with the optimal Graver step, we reduce the gap between the value of an optimal solution and our current solution by a factor of at least $1 - 1/(2nt)$ due to the pigeonhole principle. Our algorithm finds an augmenting step that is at least half as good as the optimal Graver step. Therefore the gap to the optimal solution is still reduced by at least a factor of $1 - 1/(4nt)$.

Let us now estimate the running time. We first have to compute the splitter. Proposition 6 states that this can be done in time $k^{O(1)} n \log(n) = (rs\Delta)^{O(rs)} n \log(n)$. Next, we try all necessary values for the weight λ, i. e., the values that lie between the smallest lower bound ℓ_i for some variable x_i and the largest upper bound u_j for some variable x_j with a step length of the form 2^k. Computing only these powers of 2, we get $O(\log(\Gamma))$ guesses. Recall that Γ denotes the largest difference between an upper bound and a lower bound, i. e., $\Gamma = \max_j u_j - \min_i \ell_i$. Fixing one, we have to find the best improving direction regarding each of the $((rs\Delta)^{O(rs)})^{O(1)} \log(n) = (rs\Delta)^{O(rs)} \log(n)$ partitions. In the first iteration, we perform initialization in time $k^{O(r)} \Delta^{O(r^2+s^2)} nt = (rs\Delta)^{O(r^2 s)} \Delta^{O(r^2+s^2)} nt$ by computing the gain for each possible summand for each set and setting up the data structure (the AVL trees). In each following iteration, we update the data structure and search for the optimum in time $k^{O(r)} \Delta^{O(r^2+s^2)} \log(nt) = (rs\Delta)^{O(r^2 s)} \Delta^{O(r^2+s^2)} \log(nt)$. Now it remains to bound the number ι of iterations needed to converge to an optimal solution. To obtain such a bound we calculate:

$$1 > (1 - 1/(4nt))^{\iota} |w^\top (x^* - x)|.$$

By reordering the term we get

$$\iota < \frac{-\log(|w^\top (x^* - x)|)}{\log(1 - 1/(4nt))}.$$

As $\log(1 + x) \geq 2x$ for all $x \in [-1/2, 0]$, we can bound $\log(1 - 1/(4nt))$ by $(-1/(4nt))$ and thus

$$\iota < O\left(\frac{-\log(|w^\top (x^* - x)|)}{-1/(4nt)}\right) \leq O(4nt \log(|w^\top (x^* - x)|)).$$

As the maximal difference between the current solution x and an optimal one x^* can be at most the maximal value of w times the largest number in between the bounds for each variable, we get $|w^\top (x^* - x)| \leq nt \max_i |w_i| \cdot \Gamma$ and thus

$$\begin{aligned} \iota < O(4nt \log(|w^\top (x^* - x)|)) &\leq O(nt \log(nt \max_i |w_i| \cdot \Gamma)) \\ &\leq O(nt \log(nt\Gamma \max_i |w_i|)). \end{aligned}$$

Let L denote the encoding length of the largest integer in the input. Clearly 2^L bounds the largest absolute value in w and thus, we get

$$\begin{aligned} \iota < O(nt \log(nt\Gamma \max_i |w_i|)) &= O(nt \log(nt\Gamma 2^L)) \\ &= O(nt \log(nt\Gamma 2^L)). \end{aligned}$$

Hence, after this number of steps by always improving the gain by a factor of at least $1-1/(4nt)$, we close the gap between the initial solution and an optimal one. Given this, we can now bound the overall running time with:

$$\begin{aligned}
&\underbrace{(rs\Delta)^{O(rs)}n\log(n)}_{\text{Splitter}}+\underbrace{O(\log(\Gamma))}_{\lambda\text{ Guesses}}\cdot\underbrace{(rs\Delta)^{O(rs)}\log(n)}_{\text{Partitions}}\\
&\cdot\underbrace{(rs\Delta)^{O(r^2s)}\cdot(rs\Delta)^{O(r^2+s^2)}nt}_{\text{First Iteration}}+\underbrace{O(nt\log(nt\Gamma 2^L))}_{\iota}\\
&\cdot\underbrace{O(\log(\Gamma))}_{\lambda\text{ Guesses}}\cdot\underbrace{(rs\Delta)^{O(rs)}\log(n)}_{\text{Partitions}}\cdot\underbrace{(rs\Delta)^{O(r^2s)}\cdot(rs\Delta)^{O(r^2+s^2)}\log(nt))}_{\text{Update Time}}\\
&=O((nt\log(nt\Gamma 2^L))\cdot O(\log(\Gamma))(rs\Delta)^{O(r^2s+s^2)}\log^2(nt)\\
&=(rs\Delta)^{O(r^2s+s^2)}\cdot O(\log^2(\Gamma+2^L)nt\log^3(nt)).
\end{aligned}$$

Here, *Splitter* denotes the time to compute the initial set $\mathcal{P}$ of partitions and *Partitions* denotes the cardinality of $\mathcal{P}$. *First Iteration* is the time to solve the first iteration of the (P,k)-best step problem. Further, *λ Guesses* is the number of guesses we have to do to get the right weight. Lastly, *Update Time* is the time needed to solve each following (P,k)-best step including updating the bounds and data structures.

Note that we still have to argue about finding the initial solution since in the construction of the modified n-fold ILP the parameters slightly change. The length of a brick expands to $t'=t+r+s$. This, however, can be hidden in the big O notation of $(rs\Delta)^{O(r^2s+s^2)}$. Further, Γ', the biggest difference in the upper and lower bounds can change, as we introduce new variables admitting new bounds. The difference between the bounds of old variables does not change. For the new variables, however, the difference can be as large as $\|b'\|_\infty$. Thus, we bound this value by

$$\|b'\|_\infty=\|b-\mathcal{A}\ell\|_\infty\leq\|b\|_\infty+\|\mathcal{A}\ell\|_\infty\leq\|b\|_\infty+\Delta\|\ell\|_1\leq O(\Delta nt2^L).$$

We conclude that the running time for finding an initial solution (and also the overall running time) is

$$\begin{aligned}
&(rs\Delta)^{O(r^2s+s^2)}O(\log^2(\Gamma'+2^L)nt\log^3(nt))\\
&=(rs\Delta)^{O(r^2s+s^2)}O(\log^2(\Delta 2^Lnt)nt\log^3(nt))\\
&=(rs\Delta)^{O(r^2s+s^2)}L^2nt\log^5(nt).
\end{aligned}$$

□

HANDLING INFINITE BOUNDS In the case that not all variables have finite bounds, we have to introduce artificial bounds first. Here we can proceed as in [47] where first the LP relaxation is solved to obtain an optimal fractional solution z^*. The proximity result from [47] states that if the n-fold ILP is feasible, an optimal integral solution x^* exists such that $\|x^*-z^*\|_1\leq nt(rs\Delta)^{O(rs)}$. This allows us to introduce artificial upper bounds for the unbounded variables: We shift all variables by $-\lfloor z^*\rfloor$ where $\lfloor z^*\rfloor$ is the vector z^* rounded

down in each component. More precisely, we subtract $\mathcal{A}\lfloor z^* \rfloor$ from the right-hand side and subtract $\lfloor z^* \rfloor$ from upper and lower bounds. Each solution for the shifted n-fold corresponds to one for the original problem after adding $\lfloor z^* \rfloor$. Moreover, if there is an optimal integral solution for the shifted n-fold, there is also an optimal solution x^* with $\|x^*\|_1 \leq nt(rs\Delta)^{O(rs)}$. In particular, capping all bounds at $nt(rs\Delta)^{O(rs)}$ does not affect the optimal solution.

This also improves the dependency from L^2 to L in the calculations of the previous section: The multiplicative factor $O(\log(\Gamma))$ for the guesses of λ, where Γ is the maximum distance of a lower and an upper bound, was previously bounded by $O(L\log(\Delta nt))$, but can now be bounded by $O(\log(nt(rs\Delta)^{O(rs)}))$, which is almost negligible. This yields a total running time of $(rs\Delta)^{O(r^2s+s^2)}\, Lnt\log^5(nt) + \mathrm{LP}$. However, this comes at the cost of solving the corresponding relaxation of the n-fold and solving this LP can be computationally expensive. Further, note that even if we only have bounded variables, we can use the proximity result as above to lower the dependency on L.

Theorem 11. *The dynamic augmenting step algorithm described above computes an optimal solution for the n-fold Integer Linear Programming problem in time $(rs\Delta)^{O(r^2s+s^2)}Lnt\log^5(nt) + LP$, even if some variables have infinite upper bounds. Here LP is the running time to solve the corresponding relaxation of the n-fold ILP problem.*

The current best algorithm takes time $2^{O(r^2+rs^2)}(nt)^{1+o(1)}$ to solve the n-fold ILP relaxation [39]. This yields an overall running time of $(rs\Delta)^{O(r^2s+s^2)}$ $Lnt\log^5(nt) + 2^{O(r^2+rs^2)}(nt)^{1+o(1)}$ for solving the n-fold ILP, even if some variables have infinite upper bounds. Solving the relaxation might thus dominate the running time for solving the n-fold ILP with finite upper bounds. Fortunately, we can avoid solving the LP as we describe in the following section using new structural results.

3.3 STRUCTURAL PROPERTIES OF SOLUTIONS

This section presents some structural properties of solutions for the n-fold Integer Linear Programming problem. In particular, we first prove that in an n-fold ILP, there always exists a solution of small norm (if the n-fold ILP has a finite optimum). Therefore, we can apply the algorithm for finite variable bounds by replacing every infinite one with this value. This circumvents the necessity of solving the corresponding Linear Program. Then, we present some bound on the sensitivity of an n-fold ILP. This bound is not needed in our algorithm, but we believe that it is of independent interest since it implies small sensitivity for problems that can be expressed as n-fold ILPs.

Lemma 12. *If the n-fold ILP is feasible and $\overline{y}$ is some vector satisfying the variable bounds, then there exists a feasible solution x with $\|x\|_1 \leq O(rs\Delta)^{rs+1}$ $(\|\overline{y}\|_1 + \|b\|_1)$*

Proof. For this proof, we reuse the matrix $\mathcal{A}^{\mathrm{init}}$ as defined in Section 3.2. Recall that this matrix is based on $\mathcal{A}$, but it has one column added for each

row with value 1 in this row and 0 in all others. In addition, it has some 0 columns to maintain the n-fold structure. We note that the n-fold parameters r, s and Δ remain the same and only t increases. Hence, this does not affect the bound $O(rs\Delta)^{rs}$ on the Graver element norm in Proposition 5. Let $\overline{y}$ be a vector satisfying all variable bounds of the initial n-fold. We extend it to a vector $\overline{y}'$ for $\mathcal{A}^{\text{init}}$ with $\mathcal{A}^{\text{init}} \cdot \overline{y}' = b$: The components corresponding to columns in $\mathcal{A}$ remain the same as in $\overline{y}$ and the new column with value 1 in row i is set to $(b - \mathcal{A}\overline{y})_i$. Thus the ℓ_1-norm of the new components in $\overline{y}'$ is

$$\|b - \mathcal{A}\overline{y}\|_1 \leq \|b\|_1 + \|\mathcal{A}\overline{y}\|_1 \leq \|b\|_1 + (r+s)\Delta\|\overline{y}\|_1.$$

Let x^* be a vector with $\mathcal{A}^{\text{init}} \cdot x^* = b$ that does not use any of the new columns, satisfies the variable bounds, and minimizes $\|\overline{y}' - x^*\|_1$. Since the initial n-fold is feasible, such a vector x^* must exist. We consider the decomposition into Graver elements $\sum_i \lambda_i g_i = x^* - \overline{y}'$ and assume w.l.o.g. that all $\lambda_i \neq 0$ and $g_i \neq 0$. Note that the actual number of Graver elements used for the decomposition is not important, as we bound the value of any sum independently of the number of summands occurring in it. To clarify this in the estimations, we do not write the concrete upper bounds on the sums.

Each Graver element contains a non-zero component for at least one of the new columns since otherwise, $x^* - g_i$ would be a vector closer to $\overline{y}'$ contradicting the minimality. As all Graver elements are sign-compatible and the ℓ_1-norm of the new columns in $x^* - \overline{y}'$ is at most $\|b\|_1 + (r+s)\Delta\|\overline{y}\|_1$, this term also bounds $\sum_i \lambda_i$. We conclude that

$$\begin{aligned}
\|x^*\|_1 &\leq \|\overline{y}'\|_1 + \|\sum_i \lambda_i g_i\|_1 \leq \|\overline{y}\|_1 + \|b\|_1 + (r+s)\Delta\|\overline{y}\|_1 + \sum_i \lambda_i \|g_i\|_1 \\
&\leq \|\overline{y}\|_1 + (\|b\|_1 + (r+s)\Delta\|\overline{y}\|_1)(1 + O(rs\Delta)^{rs}) \\
&\leq (\|b\|_1 + \|\overline{y}\|_1) \cdot O(rs\Delta)^{rs+1}.
\end{aligned}$$

□

Lemma 13. *If the n-fold ILP is bounded and feasible, then there exists an optimal solution x with $\|x\|_1 \leq (rs\Delta)^{O(rs)}(\|b\|_1 + nt\zeta)$ where ζ denotes the largest absolute value among all finite variable bounds.*

Proof. Clearly, there exists a (possibly infeasible) solution $\overline{y}$ satisfying the bounds with $\|\overline{y}\|_1 \leq nt\zeta$. By the previous lemma, we know that there is a feasible solution y with $\|y\|_1 \leq (rs\Delta)^{O(rs)}(\|b\|_1 + nt\zeta)$. Let x^* be an optimal solution of minimal norm. W.l.o.g. assume that $x^* - y$ has only non-negative entries. If there is a negative entry, consider the equivalent n-fold ILP with the corresponding column inverted and its bounds inverted and swapped.

We know that there is a decomposition of $x^* - y$ into weighted Graver basis elements $\sum_i \lambda_i g_i = x^* - y$. Assume w.l.o.g. that all $\lambda_i \neq 0$ and all $g_i \neq 0$. Since every g_i is sign-compatible with $x^* - y$, we have that all g_i are non-negative as well. Furthermore, it holds that $w^\top g_i > 0$ for every g_i since otherwise, $x^* - g_i$ would be a solution of smaller norm with an objective value that is not worse. Let us now construct a contradiction: Suppose that there is some

g_i where all variables corresponding to the indices in $\operatorname{supp}(g_i)$ have infinite upper bounds. Then the n-fold ILP is clearly unbounded since $y + \alpha g_i$ is feasible for every $\alpha > 0$ and in this way, we can scale the objective value beyond any bound. Thus, every Graver basis element adds at least the value 1 to some finitely bounded variable. This implies that $\sum_i \lambda_i \leq \|y\|_1 + nt\zeta$: If not, then by the pigeonhole principle there is some finitely bounded variable x^*_j with

$$x^*_j = y_j + (\sum_i \lambda_i g_i)_j > y_j + \zeta + |y_j| \geq \zeta.$$

Since x^* is feasible, this cannot be the case. We conclude

$$\begin{aligned}\|x^*\|_1 &\leq \|y\|_1 + \sum_i \lambda_i \|g_i\|_1 \leq \|y\|_1 + O(rs\Delta)^{rs} \sum_i \lambda_i \\ &\leq O(rs\Delta)^{rs}(\|y\|_1 + nt\zeta) \leq (rs\Delta)^{O(rs)}(\|b\|_1 + nt\zeta). \quad \square\end{aligned}$$

This yields an alternative approach to solving the LP relaxation because we can simply replace all infinite bounds with $\pm(rs\Delta)^{O(rs)} nt2^L$. Then, we can apply the algorithm that works only on finite variable bounds. The new encoding length L' of the largest integer in the input can be bounded by

$$L' \leq \log((rs\Delta)^{O(rs)} 2^L nt) \leq O(rs \log(rs\Delta) L \log(nt)).$$

This way we obtain the following.

Corollary 14. *We can compute an optimal solution for an n-fold ILP in time $(rs\Delta)^{O(r^2s+s^2)}\, L^2 nt \log^7(nt)$.*

In a similar way, we can derive the following bound on the sensitivity of an n-fold ILP.

Theorem 15. *Let x be an optimal solution of an n-fold ILP with right-hand side b, in particular, $\mathcal{A}x = b$. If the right hand side changes to b' and the n-fold ILP still has a finite optimum, then there exists an optimal solution x' for b' ($\mathcal{A}x' = b'$) with $\|x - x'\|_1 \leq O(rs\Delta)^{rs}\|b - b'\|_1$.*

It is notable that this bound does not depend on n. This is in contrast to the known bounds for the distance between LP and ILP solutions of an n-fold ILP [47].

Proof. Consider the matrix $\mathcal{A}^{\text{init}}$ from the construction used for finding an initial solution, that is, identity matrices are added after every block. We leave the objective function the same, that is, we obtain w^{init} by padding w with zeros on the new columns. Finally, we use b' as the right-hand side. For some y we write y_{old} and y_{new} for the vector restricted to the old variables and the variables added in the matrix $\mathcal{A}^{\text{init}}$ respectively (with all others being 0). This means $y = y_{\text{old}} + y_{\text{new}}$.

We define a solution y by setting y_{old} to x and choose y_{new} such that $\mathcal{A}^{\text{init}} \cdot y_{\text{new}} = b' - b$. Note that such a y_{new} exists, since $\mathcal{A}^{\text{init}}$ is the identity

matrix plus some zero columns when restricted to the new variables. Moreover, choose a solution y' with $\mathcal{A}^{\text{init}} \cdot y'_{\text{new}} = 0$ that maximizes $(w^{\text{init}})^\top y'$. Furthermore, we assume that y' is chosen so as to minimize $\|y - y'\|_1$. Let $\sum_i \lambda_i g_i = y' - y$ be the decomposition of the difference into Graver basis elements and assume w.l.o.g. that all $\lambda_i \neq 0$ and all $g_i \neq 0$. Suppose toward contradiction that there is some g_i where all of $\operatorname{supp}(g_i)$ correspond to old variables. If $(w^{\text{init}})^\top g_i > 0$ then $(w^{\text{init}})^\top (y + g_i)_{\text{old}} = (w^{\text{init}})^\top (y + g_i) > (w^{\text{init}})^\top y = w^\top x$ and $\mathcal{A}^{\text{init}} \cdot (y + g_i)_{\text{old}} = \mathcal{A}^{\text{init}} \cdot (y + g_i) - \mathcal{A}^{\text{init}} \cdot y_{\text{new}} = b$. Hence, after omitting the new variables, $y + g_i$ is a better solution than x, a contradiction. If, on the other hand, $(w^{\text{init}})^\top g_i \leq 0$ then $y' - g_i$ is a solution of at least the same value, as y' and thus $\|y - y'\|_1$ is not minimal. Indeed, this means $\|(g_i)_{\text{new}}\|_1 \geq 1$ for all g_i. In other words, each Graver element contains a non-zero new variable. Due to the sign-compatibility of these Graver elements, we get

$$\begin{aligned}\sum_i \lambda_i \leq \|(\sum_i \lambda_i g_i)_{\text{new}}\|_1 &= \|\mathcal{A}^{\text{init}} \cdot (\sum_i \lambda_i g_i)_{\text{new}}\|_1 \\ &= \|\mathcal{A}^{\text{init}} \cdot (y' - y)_{\text{new}}\|_1 = \|b - b'\|_1.\end{aligned}$$

Therefore,

$$\|y - y'\|_1 = \|\sum_i \lambda_i g_i\|_1 \leq O(rs\Delta)^{rs} \sum_i \lambda_i \leq O(rs\Delta)^{rs} \|b - b'\|_1.$$

Finally, we obtain x' by omitting the new variables in y', which implies $\|x - x'\|_1 \leq \|y - y'\| \leq O(rs\Delta)^{rs} \|b - b'\|_1$. □

I don't pretend to see the path, but I know it's there all the same. One day, we'll look back and wonder how we ever missed it.

— from *The Warded Man* by Peter V. Brett

4

HARDNESS FOR 2-STAGE STOCHASTIC ILPS

The previous chapter presents an algorithm to solve n-fold ILPs efficiently. In particular, we give an algorithm that runs in near-linear time with respect to the number of columns and single-exponentially in the number of rows of a single block (and the largest absolute value Δ appears in that basis). In this chapter, we investigate the closely related 2-stage stochastic ILPs, whose constraint matrix is the transpose of the n-fold one. These integer linear programs arise from the 4-block n-fold ILPs by setting $r = 0$. Thus, the constraint matrix admits a block-structure where non-zero entries only appear in the first few columns and block-wise along the the diagonal beside, yielding:

$$\mathcal{A} = \begin{pmatrix} D_1 & B_1 & 0 & \dots & 0 \\ D_2 & 0 & B_2 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ D_n & 0 & \dots & 0 & B_n \end{pmatrix}.$$

Recall that $D_1, \dots, D_n \in \mathbb{Z}^{s \times q}$ and $B_1, \dots, B_n \in \mathbb{Z}^{s \times t}$ are integer matrices themselves. As before, Δ denotes the largest absolute entry in $\mathcal{A}$, b the right-hand side, w the cost vector and u, ℓ the upper and lower bounds on the variables.

Such 2-stage stochastic ILPs are a common tool in stochastic programming [107] and they are often used in practice to model uncertainty of decision making over time. Due to their applicability, a lot of research has been done in order to solve these (mixed) ILPs efficiently in practice. Since we focus on the theoretical aspects of 2-stage stochastic ILPs in this chapter, we only refer the reader to the surveys [67, 120, 150] regarding the practical methods.

The current state-of-the-art algorithms to solve 2-stage stochastic ILPs admit a running time of $nt \log^3(nt) \cdot |I| \cdot 2^{(2\Delta)^{s^2q+q^2s}}$ where $|I|$ is the binary encoding length of the problem [48] or respectively of $n \log^{O(qt)}(n) 2^{(2\Delta)^{O(q^2+qt)}}$ [40] by a recent result transferring the idea for n-fold ILPs from the work [39]. The first result in that respect was by Hemmecke and Schulz [85] who provided an algorithm with a running time of $f(s,q,t,\Delta) \cdot \text{poly}(n)$ for some computable function f. However, due to the use of an existential result from commutative algebra, no explicit bound could be stated for f. Klein showed a running time of $O(n^2t^2 \cdot |I| \cdot \log^2(nt) \cdot (sq\Delta)^{sq^2(2s\Delta+1)^{sq^2}})$ [114].

The latest algorithms raise the question whether 2-stage stochastic ILPs are indeed intrinsically harder to solve than the closely related n-fold ILPs. We answer this question affirmatively by showing a double-exponential lower bound on the running time for any algorithm solving the 2-STAGE STOCHASTIC INTEGER LINEAR PROGRAMMING (2-STAGE ILP) problem. Here, the 2-STAGE ILP problem is the corresponding decision variant which asks whether the ILP admits a feasible solution.

To prove this hardness, we reduce from the QUADRATIC CONGRUENCES problem. This problem asks whether there exists a number $z \leq \gamma$ such that $z^2 \equiv \alpha \mod \beta$ for some $\gamma, \alpha, \beta \in \mathbb{N}$. This problem was proven to be NP-hard by Manders and Adleman [128] already in 1978 by showing a reduction from 3-SAT. This hardness even persists if the prime factorization of β is given [128]. By this result, Manders and Adleman proved that it is NP-complete to compute the solutions of diophantine equations of degree 2. However, their reduction yields large parameters. In particular, the occurrences of each prime factor in the prime factorization of β is too large to obtain the desired lower bound for the 2-STAGE ILP problem. The occurrence of each prime factor is at least linear in the number of variables and clauses of the underlying 3-SAT problem. We give a new reduction yielding a stronger statement: The QUADRATIC CONGRUENCES problem is NP-hard even if the prime factorization of β is given and each prime factor occurs at most once (except 2 which occurs four times). Beside being useful to prove the lower bounds for solving 2-stage stochastic ILPs, we think this result is of independent interest. We obtain a novel structure which may be helpful in various related problems, or may yield stronger statements of past results which used the QUADRATIC CONGRUENCES problem.

In order to achieve the desired lower bounds on the running time, we make use of the Exponential Time Hypothesis (ETH) – a widely believed conjecture stating that the 3-SAT problem cannot be solved in subexponential time with respect to the number of variables:

Conjecture 1 (ETH [91]). *The 3-SAT problem cannot be solved in time less than $O(2^{\delta_3 n_3})$ for some constant $\delta_3 > 0$ where n_3 is the number of variables in the instance.*

Note that we use the index 3 for all variables of the 3-SAT problem.

Using the ETH, plenty lower bounds for various problems are shown, for an overview on the techniques and results see e.g. [41]. So far, the best algorithm runs in time $O(2^{0.387 n_3})$, i. e., it follows that $\delta_3 \leq 0.387$ [41].

We also need the Chinese Remainder theorem (CRT) for some of the proofs, which states the following:

Proposition 16 (CRT [93]). *Let $n_1, \ldots, n_k$ be pairwise co-prime numbers. Further, let $i_1, \ldots, i_k$ be some integers. Then there exists integers x satisfying $x \equiv i_j \mod n_j$ for all j. Further, any two solutions x_1, x_2 are congruent modulo $\prod_{j=1}^{k} n_j$.*

SUMMARY OF RESULTS

- We give a new reduction from the 3-SAT problem to the Quadratic Congruences problem which proves a stronger NP-hardness result: The Quadratic Congruences problem remains NP-hard even if the prime factorization of β is given and each prime number greater than 2 occurs at most once and the prime number 2 occurs four times. This does not follow from the original proof. In contrast, the original proof generates each prime factor at least $O(n_3 + m_3)$ times, where m_3 is the number of clauses in the formula. Our reduction circumvents this necessity, yet neither introduces noteworthily more nor larger prime factors. The proof is based on the original one. We believe this result is of independent interest.

- Based on this new reduction, we show strong NP-hardness for the so-called Non-Unique Remainder problem. In this algorithmic number theoretical problem, we are given $x_1, \ldots, x_{n_{\text{NR}}}, y_1, \ldots, y_{n_{\text{NR}}}, \zeta \in \mathbb{N}$ and pairwise coprime numbers $q_1, \ldots, q_{n_{\text{NR}}}$. The question is to decide whether there exists a number $z \in \mathbb{Z}_{>0}$ with $z \leq \zeta$ satisfying the following congruences:

$$\begin{aligned} &z \bmod q_1 \in \{x_1, y_1\} \\ &z \bmod q_2 \in \{x_2, y_2\} \\ &\vdots \\ &z \bmod q_{n_{\text{NR}}} \in \{x_{n_{\text{NR}}}, y_{n_{\text{NR}}}\}. \end{aligned}$$

  In other words, either the residue x_i or y_i should be met for each equation. This problem is a natural generalization of the Chinese Remainder theorem where $x_i = y_i$ for all i. In that case, however, the problem can be solved on polynomial time using the Extended Euclidean algorithm. To the best of our knowledge, the Non-Unique Remainder problem has not been considered in the literature so far.

- Finally, we show that the Non-Unique Remainder problem can be modeled by a 2-stage stochastic ILP. Assuming the ETH, we can then conclude a doubly exponential lower bound of $2^{2^{\delta(s+t)}}|I|^{O(1)}$ on the running time for any algorithm solving 2-stage stochastic ILPs. The double exponential lower bound even holds if $q = 1$ and $\Delta, \|b\|_\infty \in O(1)$. This proves the suspicion that 2-stage stochastic ILPs are significantly harder to solve than n-fold ILPs with respect to the dimensions of the block matrices and Δ. Furthermore, it implies that the current state-of-the-art algorithms [40, 48] for solving 2-stage stochastic ILPs are indeed (nearly) optimal with respect to this parameters.

FURTHER RELATED WORK As mentioned above, n-fold ILPs are closely related to the 2-stage ILP problem. In recent years, there was a significant progress in the development of algorithms for n-fold ILPs and their lower

bounds respectively, see the previous chapter for details. The best known, strongly polynomial algorithm to solve these n-fold ILPs has a running time of $2^{O(rs^2)}(rs\Delta)^{O(r^2s+s^2)}(nt)^{1+o(1)}$ [39] while the best known lower bound is $\Delta^{\delta_{n\text{-fold}}(r+s)^2}$ for some $\delta_{n\text{-fold}} > 0$ [48].

Despite their similarity, it seems that 2-stage stochastic ILPs are significantly harder to solve than n-fold ILPs. Yet, no superexponential lower bound for the running time of any algorithm solving the 2-STAGE ILP problem was shown. There is a lower bound for a more general class of ILPs in [48] that contain 2-stage stochastic ILPs showing that the running time is double-exponential parameterized by the topological height of the treedepth decomposition of the primal or dual graph. However, the topological height of 2-stage stochastic ILPs is constant and thus, no strong lower bound can be derived for this case.

If we relax the necessity of an integral solution, the 2-stage stochastic LP problem becomes solvable in time $2^{2\Delta^{O(s^3)}} n \log^3(n) \log(\|u-\ell\|_\infty) \log(\|w\|_\infty)$ [23]. For the case of mixed integer linear programs there exists an algorithm solving 2-stage stochastic MILPs in time $2^{\Delta^{\Delta^{s^{O(s^2)}}}} n \log^3(n) \log(\|u-\ell\|_\infty) \log(\|w\|_\infty)$ [23]. Both results rely on the fractionality of a solution, whose size is only dependent on the parameters. This allows us to scale the problem such that it basically becomes an ILP (as the solution has to be integral) and thus, state-of-the-art algorithms for 2-stage stochastic ILPs can be applied.

Regarding the general case of 4-block n-fold ILPs, only little is known: They are in XP [82]. Further, a lower and upper bound on the Graver Basis elements (inclusion-wise minimal kernel elements) of $O(n^r)f(q,s,t,\Delta)$ was shown recently [31].

STRUCTURE OF THIS CHAPTER Section 4.1 presents the stronger hardness result for the QUADRATIC CONGRUENCES problem we derive by giving a new reduction from the 3-SAT problem. Then, we show that the QUADRATIC CONGRUENCES problem can be modeled as a 2-stage stochastic ILP in Section 4.2. To do so, we introduce a new problem called the NON-UNIQUE REMAINDER problem as an intermediate step during the reduction. Finally, in Section 4.3, we bring the reductions together to prove the desired lower bound. This involves a construction which lowers the absolute value of Δ at the cost of slightly larger block dimensions.

4.1 ADVANCED HARDNESS FOR QUADRATIC CONGRUENCES

This section proves that every instance of the 3-SAT problem can be transformed into an equivalent instance of the QUADRATIC CONGRUENCES problem in polynomial time. Recall that given the numbers $\alpha, \beta, \gamma \in \mathbb{Z}$, the QUADRATIC CONGRUENCES problem asks whether there exists a number $z \leq \gamma$ such that $z^2 \equiv \alpha \mod \beta$ holds. This problem was proven to be NP-hard by Manders and Adleman [128] showing a reduction from 3-SAT. This hardness even

persists when the prime factorization of β is given [128]. However, we aim for an even stronger statement: The Quadratic Congruences problem remains NP-hard even if the prime factorization of β is given and each prime number greater than 2 occurs at most once and the prime number 2 occurs four times. This does not follow from the original hardness proof. In contrast, if n_3 is the number of variables and m_3 the number of clauses in the 3-SAT formula then β admits a prime factorization with $O(n_3+m_3)$ different prime numbers each with a multiplicity of at least $O(n_3+m_3)$. Even though our new reduction lowers the occurrence of each prime factor greatly, we neither introduces noteworthily more nor larger prime factors.

While the structure of our proof resembles the original one from [128], adapting it to our needs requires various new observations concerning the behaviour of the newly generated prime factors and the functions we introduce. The original proof heavily depends on the numbers being high powers of the prime factors, whereas we employ careful combinations of (new) prime factors. This requires us to introduce other number theoretical results into the arguments.

In the following, before presenting the reduction and showing its correctness formally, we want to give an idea of the hardness proof. The reduction may seem non-intuitive at first, as it only shows the final result of equivalent transformations between various problems until we reach the Quadratic Congruences one. In the following, we list all these problems in order of their appearance whose strong NP-hardness is shown implicitly along the way. Afterwards, we give short ideas of their respective equivalence, which is then proved formally in separate claims in the next theorem. Note that not all variables are declared at this point, but are also not necessary to understand the proof sketch.

- (3-SAT) Is there a truth assignment η that satisfies all clauses σ_k of the 3-SAT formula Φ simultaneously?
- (P2) Are there values $y_k \in \{0,1,2,3\}$ and a truth assignment η such that $0 = y_k - \sum_{x_i \in \sigma_k} \eta(x_i) - \sum_{\bar{x_i} \in \sigma_k} (1-\eta(x_i)) + 1$ for all k?
- (P3) Are there values $\alpha_j \in \{-1,+1\}$ such that $\sum_{j=0}^{\nu} \theta_j \alpha_j \equiv \tau \bmod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i$ for some θ_j, some τ specified in dependence on the formula later on, and some prime numbers p_i and p^*?
- (P5) Is there an $x \in \mathbb{Z}$ satisfying

$$0 \leq |x| \leq H \tag{P5.1}$$

$$x \equiv \tau \bmod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i \tag{P5.2}$$

$$(H+x)(H-x) \equiv 0 \bmod K \tag{P5.3}$$

  for some H dependent on the θ_j and K being a product of primes?

- (P6) Is there an $x \in \mathbb{Z}$ satisfying

$$0 \leq |x| \leq H \tag{P6.1}$$

$$(\tau - x)(\tau + x) \equiv 0 \bmod 2^4 \cdot p^* \prod_{i=1}^{m'} p_i \tag{P6.2}$$

$$(H + x)(H - x) \equiv 0 \bmod K? \tag{P6.3}$$

- (Quadratic Congruences) Is there a number $x \leq H$ such that $(2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i + K)x^2 \equiv K\tau^2 + 2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i H^2 \bmod 2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i \cdot K$?

The 3-SAT problem is transformed to Problem (P2) by using the straightforward interpretation of truth values as numbers 0 and 1 and the satisfiability of a clause as the sum of its literals being larger zero. Introducing slack variables y_k yields the above form, see Claim 19.

Multiplying each equation of (P2) with growing factors and then forming their sum preserves the equivalence of these systems. Introducing some modulo consisting of unique prime factors larger than the outcome of the largest possible sum obviously does not influence the system. Replacing the variables $\eta(x_i)$ and y_k by variables α_j with domain $\{-1, +1\}$, re-arranging the term and defining parts of the formula as the variables θ_j yields Problem (P3), see Claim 20.

We then introduce some Problem (P4) to integrate the condition $x \leq H$. The problem asks whether there exists some $x \in \mathbb{Z}$ such that

$$0 \leq |x| \leq H \tag{P4.1}$$

$$(H + x)(H - x) \equiv 0 \bmod K? \tag{P4.2}$$

By showing that each solution to the system (P4) is of form $\sum_{j=0}^{\nu} \theta_j \alpha_j$, we can combine (P3) and (P4) yielding (P5), see Claim 21.

Using some observations about the form of solutions to Problem (P5), we can re-formulate it as Problem (P6), see Claim 22.

Next, we use the fact that $p^* \prod_{i=1}^{m'} p_i$ and K are co-prime per definition and thus, we can combine (P6.2) and (P6.3) to one equivalent equation. Using a little re-arranging this finally yields the desired Quadratic Congruences problem, see Claim 23.

Before we finally present the reduction, we first prove a lemma about the size of the product of prime numbers, which comes in handy in the respective theorem.

Lemma 17. *Denote by q_i the ith prime number. The product of the first k prime numbers $\prod_{i=1}^{k} q_i$ is bounded by $2^{2k\log(k)}$ for all $k \geq 2$.*

Proof. Denote by $\Psi(x)$ the number of prime numbers of size at most x. It holds that $\Psi(x) > x/\log(x)$ for $x \geq 17$ [146]. Setting $x = y^2$, it holds that $\Psi(y^2) > y^2/\log(y^2)$ for $y \geq 5$. As $y^2/\log(y^2) = y^2/(2\log(y)) \geq y^2/y = y$ for $y \geq 5$, it also holds that $\Psi(y^2) > y$ for $y \geq 5$. Thus, $p_i < i^2$ for $i \geq 5$, as we have at least i many prime numbers in the interval $[1, i^2]$.

Manually checking the values for the first four prime numbers shows that the equation $p_i \leq i^2$ also holds for all prime numbers greater 2. For $p_1 = 2 > 1^2$, we can simply multiply an additional factor of 2. Altogether, we can thus estimate the product of the first k prime numbers for $k \geq 2$ as

$$\begin{aligned}\prod_{i=1}^{k} q_i &\leq \prod_{i=1}^{k} (i^2) \cdot 2 = (\prod_{i=1}^{k} i)^2 \cdot 2 = (k!)^2 \cdot 2 \leq (2(k/2)^k)^2 \cdot 2 \\ &= 2^2((k/2)^k)^2 \cdot 2 = 2^3(k/2)^{2k} = 2^3 2^{2k\log(k/2)} \leq 2^{2k\log(k)}\end{aligned}$$

proving the statement. Note that $k \geq 2$ has to hold for the last estimation. □

Theorem 18. *The* Quadratic Congruences *problem is NP-hard even if the prime factorization of β is given and each prime factor greater than 2 occurs at most once and the prime factor 2 occurs 4 times.*

Proof. We show a reduction from the well-known NP-hard problem 3-SAT where we are given a 3-SAT formula Φ with n_3 variables and m_3 clauses.

Transformation: First, eliminate duplicate clauses from Φ and those where some variable x_i and its negation $\bar{x_i}$ appear together. Call the resulting formula Φ', the number of occurring variables n' and denote by m' the number of appearing clauses respectively. Let $\Sigma = (\sigma_1, \ldots, \sigma_{m'})$ be some enumeration of the clauses. Denote by $p_0, \ldots, p_{2m'}$ the first $2m' + 1$ prime numbers. Compute

$$\tau_{\Phi'} = -\sum_{i=1}^{m'} \prod_{j=1}^{i} p_j.$$

Further, compute for each $i \in 1, 2, \ldots, n'$:

$$f_i^+ = \sum_{x_i \in \sigma_j} \prod_{k=1}^{j} p_k \quad \text{and} \quad f_i^- = \sum_{\bar{x_i} \in \sigma_j} \prod_{k=1}^{j} p_k.$$

Set $\nu = 2m' + n'$. Compute the coefficients c_j for all $j = 0, 1, \ldots, \nu$ as follows: Set $c_0 = 0$. For $j = 1, \ldots, 2m'$ set

$$c_j = -\frac{1}{2} \prod_{i=1}^{j} p_i \quad \text{for } j = 2k-1 \quad \text{and} \quad c_j = -\prod_{i=1}^{j} p_i \quad \text{for } j = 2k.$$

Compute the remaining ones for $j = 1, \ldots n'$ as $c_{2m'+j} = 1/2 \cdot (f_j^+ - f_j^-)$. Further, set $\tau = \tau_{\Phi'} + \sum_{j=0}^{\nu} c_j + \sum_{i=1}^{n'} f_i^-$.

Denote by $q_1, \ldots, q_{\nu^2+2\nu+1}$ the first $\nu^2 + 2\nu + 1$ prime numbers. Let $p_{0,0}, p_{0,1}, \ldots, p_{0,\nu}, p_{1,0}, \ldots, p_{\nu,\nu}$ be the first $(\nu+1)^2 = \nu^2 + 2\nu + 1$ prime numbers greater than $(4(\nu+1)2^3 \prod_{i=1}^{\nu^2+2\nu+1} q_i)^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))}$ and greater than $p_{2m'}$. Define p^* as the $(\nu^2+2\nu+2m'+13)$th prime number.

Determine the parameters θ_j for $j = 0, 1, \ldots, \nu$ as the least θ_j satisfying:

$$\theta_j \equiv c_j \bmod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i,$$
$$\theta_j \equiv 0 \bmod \prod_{i=0, i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k},$$
$$\theta_j \not\equiv 0 \bmod p_{j,1}.$$

Set the following parameters:

$$H = \sum_{j=0}^{\nu} \theta_j \text{ and } K = \prod_{i=0}^{\nu} \prod_{k=0}^{\nu} p_{i,j}.$$

Finally, set

$$\alpha = (2^4 \cdot p^* \prod_{i=1}^{m'} p_i + K)^{-1} \cdot (K\tau^2 + 2^4 \cdot p^* \prod_{i=1}^{m'} p_i \cdot H^2),$$
$$\beta = 2^4 \cdot p^* \prod_{i=1}^{m'} p_i \cdot K,$$
$$\gamma = H.$$

where $(2^4 \cdot p^* \prod_{i=1}^{m'} p_i + K)^{-1}$ is the inverse of $(2^4 \cdot p^* \prod_{i=1}^{m'} p_i + K) \bmod 2^4 \cdot p^* \prod_{i=1}^{m'} p_i \cdot K$.

Correctness: We show that the satisfiability of the formula Φ is equivalent to a multitude of (systems of) equations, i. e., the formula has a satisfying truth assignment on the variables if and only if the (systems of) equations admit a solution. By this, we prove the hardness for various problems along the way. These are listed above with their respective equivalence sketched. In the following, we separate each of these steps by claims.

However, before we start with the transformations of the formula, we first observe some properties about the generated prime factors. These come in handy for the estimations later on. In particular, we want to show that choosing p^* as the $(\nu^2 + 2\nu + 2m' + 13)$th prime factor satisfies $p^* > p_{\nu,\nu}$: Suppose $p_{2m'} \geq (4(\nu+1)2^3 \cdot \prod_{i=1}^{\nu^2+2\nu+1} q_i)^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))}$. Then $p_{\nu,\nu}$ is the $(\nu^2 + 2\nu + 1 + 2m' + 1)$th prime number and thus $p^* > p_{\nu,\nu}$.

Otherwise, if $p_{2m'} < (4(\nu+1)2^3 \prod_{i=1}^{\nu^2+2\nu+1} q_i)^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))}$, we bound the function values as follows:

$$\begin{aligned}
&(4(\nu+1)2^3 \prod_{i=1}^{\nu^2+2\nu+1} q_i)^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))} \\
&= 4^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))} (\nu+1)^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))} \\
&\quad \cdot (2^3)^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))} \\
&\quad \cdot (\prod_{i=1}^{\nu^2+2\nu+1} q_i)^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))} \\
&\leq 2 \cdot 2 \cdot 2 \cdot (2^{2(\nu^2+2\nu+1)\log(\nu^2+2\nu+1)})^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))} \\
&\leq 8 \cdot (4^{(\nu^2+2\nu+1)\log(\nu^2+2\nu+1)})^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))} \\
&= 8 \cdot 4 = 32.
\end{aligned}$$

The second transformation holds, as the product of the first k prime numbers is bounded by $2^{2k\log(k)}$ (for $k \geq 2$, which obviously holds here), see Lemma 17. There are 11 prime numbers in the interval $[1,32]$. Thus, $p_{\nu,\nu}$ is at most the $(11+\nu^2+2\nu+1)$th prime number and thus $p^* > p_{\nu,\nu}$.

Further, note that $p^* \leq \prod_{i=m'+1}^{\nu^2+2\nu+1} q_i$: We can bound the value of the product from beneath as $\prod_{i=m'+1}^{\nu^2+2\nu+1} q_i \geq q_{m'+1}^{\nu^2+\nu}$. Estimating the value for p^*, we use that the value of the next prime number after a number ρ is at most 2ρ [1]. Thus, as there are $\nu^2+2\nu+m'+11$ prime numbers between $p_{m'+1}$ and p^*, we get $p^* \leq q_{m'+1} \cdot 2^{\nu^2+2\nu+m'+11} \leq q_{m'+1} \cdot 2^{\nu^2+3\nu+11}$ since per definition $\nu \geq m'$ holds. Dividing both sides of the estimation by $q_{m'+1}$, it thus remains to show that $2^{\nu^2+3\nu+11} \leq q_{m'+1}^{\nu^2+\nu-1}$. Obviously, $q_{m'+1}^{\nu^2+\nu-1}$ grows for larger values of m'. The smallest reasonable value for $m' = 2$ and thus $q_{m'+1} \geq 5$. By that, we get that

$$q_{m'+1}^{\nu^2+\nu-1} \geq 5^{\nu^2+\nu-1} \geq 2^{2(\nu^2+\nu-2)} = 2^{2\nu^2+2\nu-1} \geq 2^{\nu^2+3\nu+11}$$

for all $\nu \geq 5$ and thus for all reasonable values of ν, showing the statement.

Let us now focus on the transformations of the formula Φ yielding the first claim:

Claim 19. *The 3-SAT problem asking whether there is a truth assignment η that satisfies all clauses σ_k of the 3-SAT formula Φ simultaneously is a yes-instance if and only if Problem (P2) asking whether there are values $y_k \in \{0,1,2,3\}$ and a truth assignment η such that $0 = y_k - \sum_{x_i \in \sigma_k} \eta(x_i) - \sum_{\bar{x}_i \in \sigma_k} (1-\eta(x_i)) + 1$ for all k is a yes-instance.*

Proof. Obviously, the reduced formula Φ' is satisfiable if and only if Φ is. The formula Φ' is satisfiable if there exists a truth assignment $\eta\colon \{x_1,\dots,x_{n'}\} \to \{0,1\}$ assigning a logical value to each variable $x_1,\dots,x_{n'}$ which satisfies all clauses $\sigma_1,\dots,\sigma_{m'}$ simultaneously. This can be re-written to the following equation for each clause $\sigma_k \in \Phi_k$:

$$0 = R_k = y_k - \sum_{x_i \in \sigma_k} \eta(x_i) - \sum_{\bar{x}_i \in \sigma_k} (1-\eta(x_i)) + 1, \;\; y_k \in \{0,1,2,3\}.$$

For a clause σ_k, this equation is only satisfiable if at least one variable $x_i \in \sigma_k$ has value $\eta(x_i) = 1$ or one variable occurring in its negation $\bar{x_i} \in \sigma_k$ has value $\eta(x_i) = 0$. Otherwise, we have to set $y_k = -1$ which is not allowed. □

Note that we never have to set $y_k = 3$ to satisfy the formula nor does it fulfill not satisfiable formulas. However, we allow this value, as it comes in handy later on when transforming the equation. Further, set $0 = R_0 = \alpha_0 + 1$ for $\alpha_0 \in \{-1, +1\}$ for later convenience. Clearly, the new equation is satisfiable.

Claim 20. *The Problem (P2) asking whether there are values $y_k \in \{0,1,2,3\}$ and a truth assignment η such that $0 = R_k = y_k - \sum_{x_i \in \sigma_k} \eta(x_i) - \sum_{\bar{x_i} \in \sigma_k} (1 - \eta(x_i)) + 1$ for all k is a yes-instance if and only if Problem (P3) asking whether there are values $\alpha_j \in \{-1, +1\}$ such that $\sum_{j=0}^{\nu} \theta_j \alpha_j \equiv \tau \mod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i$ is a yes-instance.*

Proof. We can bound the values of R_k for $k \in \{0, 1, \ldots, m'\}$ by $-2 \leq R_k \leq 4$. For the lower bound, the values are given by $y_k = 0$, all $x_i \in \sigma_k$ have value $\eta(x_i) = 1$ and all $\bar{x_i} \in \sigma_k$ have value $\eta(x_i) = 0$. For the upper bound, we set $y_k = 3$, all $x_i \in \sigma_k$ to $\eta(x_i) = 0$ and $\bar{x_i} \in \sigma_k$ to $\eta(x_i) = 1$. For R_0 obviously $0 \leq R_k \leq 2$ holds. Thus,

$$R_k = 0, \forall k \in \{0, 1, \ldots, m'\} \Leftrightarrow \sum_{k=0}^{m'} R_k \prod_{i=0}^{k} p_i = 0$$

as the sum is zero if all $R_k = 0$. For the opposite direction, if the sum is zero, then no $R_k \neq 0$ as the product of the prime numbers grows too fast. Thus, the other summands cannot compensate for some $R_k \neq 0$. We can bound the expression further by

$$\left|\sum_{k=0}^{m'} R_k \prod_{i=0}^{k} p_i\right| \leq 4 \sum_{k=0}^{m'} \prod_{i=0}^{k} p_i \leq 4(m'+1) \prod_{i=0}^{m'} p_i < 2^3 \cdot p^* \prod_{i=1}^{m'} p_i$$

as $p^* > p_{\nu,\nu} > p_{m'} > m' + 1$. This yields

$$R_k = 0, \forall k \in \{0, 1, \ldots, m'\} \Leftrightarrow \sum_{k=0}^{m'} R_k \prod_{i=0}^{k} p_i \equiv 0 \mod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i \quad \text{(I)}$$

as the modulo has no impact on the satisfiability of the equation.

Next, we aim to rewrite R_k by replacing the variables y_k and $\eta(x_i)$ with new variables α_j admitting a domain of $\{-1, 1\}$:

$$y_k = 1/2 \cdot [(1 - \alpha_{2k-1}) + 2 \cdot (1 - \alpha_{2k})], \; k \in \{1, \ldots, m'\},$$
$$\eta(x_i) = 1/2 \cdot (1 - \alpha_{2m'+i}), \; i \in \{1, \ldots, n'\}.$$

Obviously, the value domains of y_k and $\eta(x_i)$ are preserved. Substituting the variables and re-arranging the equation (I) yields

$$\sum_{j=0}^{\nu} c_j \alpha_j \equiv \tau \mod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i, \; \alpha_j \in \{-1, +1\}.$$

By definition of θ_j, this is equivalent to

$$\sum_{j=0}^{\nu} \theta_j \alpha_j \equiv \tau \bmod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i,\ \alpha_j \in \{-1,+1\}$$

proving the claim. □

Let $H = \sum_{j=0}^{\nu} \theta_j$ and $K = \prod_{i=0}^{\nu} \prod_{j=0}^{\nu} p_{i,j}$ be defined as before. Consider the following system where we aim to find some $x \in \mathbb{Z}$ such that:

$$0 \leq |x| \leq H \tag{P4.1}$$
$$(H+x)(H-x) \equiv 0 \bmod K \tag{P4.2}$$

We use this system to integrate the condition $x \leq H$ into the transformations. In the following, we prove that each solution of this system is of form $x = \sum_{j=0}^{\nu} \alpha_j \theta_j$ and thus Problem (P4) can be combined with Problem (P3) yielding Problem (P5).

Claim 21. *The Problem (P3) asking whether there are values $\alpha_j \in \{-1,+1\}$ such that $\sum_{j=0}^{\nu} \theta_j \alpha_j \equiv \tau \bmod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i$ is a yes-instance if and only if the Problem (P5) is a yes-instance.*

Proof. The unique solutions x to the given system (P4) are of form

$$x = \sum_{j=0}^{\nu} \alpha_j \theta_j,\ \alpha \in \{-1,+1\},\ j = 0,1,\ldots,\nu.$$

Let us start with verifying that an x of such form solves the system. First,

$$|x| = |\sum_{j=0}^{\nu} \alpha_j \theta_j| \leq \sum_{j=0}^{\nu} \theta_j = H$$

satisfies (P4.1). Further, we have that each summand in the expanded formula $(H+x)(H-x)$ has to contain all prime factors $p_{i,j}$ for $i = 0,1,\ldots,\nu$ and $j = 0,1,\ldots,\nu$ in its prime factorization to satisfy (P4.2). For $(H+x) = (\sum_{j=0}^{n} \theta_j + \sum_{j=0}^{n} \theta_j \alpha_j)$ it holds that each θ_j occurs twice where $\alpha_j = +1$, while each θ_j is canceled out by H where $\alpha_j = -1$. The other way around holds for $(H-x)$. Thus, expanding the brackets yields that each summand is a product of some θ_j and θ_k where $\alpha_j = +1$ and $\alpha_k = -1$. This implies that $j \neq k$. As each θ_j contains all prime factors of K except $p_{j,0},\ldots,p_{j,\nu}$, the product of two different θ_j and θ_k contains each prime factor occurring in K satisfying (P4.2).

Regarding the uniqueness, observe that

$$(H+x)(H-x) \equiv 0 \bmod \prod_{j=0}^{\nu} p_{i,j},\ \forall i = 0,1,\ldots,\nu.$$

Assume there exists some number $\tilde{p} = \prod_{j=0}^{\nu} p_{i,j}$ for some $i \in \{0,1,\ldots,\nu\}$ which divides $(H+x)$ and $(H-x)$ (without remainder). Thus, $(H+x) +$

$(H-x) \equiv 0 \bmod \tilde{p} \Leftrightarrow 2H \equiv 0 \bmod \tilde{p}$. As $\tilde{p}$ is a product of prime numbers greater than 2 is follows that $H \equiv 0 \bmod \tilde{p} \Leftrightarrow \sum_{j=0}^{\nu} \theta_j \equiv 0 \bmod \tilde{p}$. However, from the definition of θ_j (third condition), it follows that for each j there exist different prime numbers not present in the prime factorization of θ_j contradicting the assumption. Thus, $\tilde{p}$ divides either $(H+x)$ or $(H-x)$ (without remainder). Define

$$\alpha_i = \begin{cases} +1 & \text{if } (H-x) \equiv 0 \bmod \prod_{j=0}^{\nu} p_{i,j} \\ -1 & \text{if } (H+x) \equiv 0 \bmod \prod_{j=0}^{\nu} p_{i,j} \end{cases}$$

$$x' = \sum_{i=0}^{\nu} \alpha_i \theta_i.$$

In the following, we show that $x' \equiv x \bmod \prod_{j=0}^{\nu} p_{i,j}$ holds:

$$\begin{aligned} & x' \equiv x \bmod \prod_{j=0}^{\nu} p_{i,j} \\ \Leftrightarrow & \sum_{i=0}^{\nu} \alpha_i \theta_i \equiv x \bmod \prod_{j=0}^{\nu} p_{i,j} \\ \Leftrightarrow & \alpha_i \theta_i \equiv x \bmod \prod_{j=0}^{\nu} p_{i,j} \\ \\ \Leftrightarrow & \sum_{k=0}^{\nu} \alpha_i \theta_k \equiv x \bmod \prod_{j=0}^{\nu} p_{i,j} \\ \Leftrightarrow & \alpha_i \sum_{k=0}^{\nu} \theta_k \equiv x \bmod \prod_{j=0}^{\nu} p_{i,j} \\ \Leftrightarrow & \alpha_i H \equiv x \bmod \prod_{j=0}^{\nu} p_{i,j} \end{aligned}$$

for all $i \in \{0,1,\dots,\nu\}$. The first transformation simply inserts the definition of x'. Due to the definition of the θ_k, only the summand θ_i remains after calculating the modulo. Thus, we can sum up all θ_k with arbitrary sign, as they equal zero after calculating the modulo. In the last step, we insert the definition of H. Now, we either have $\alpha_j = +1$. Then, $H \equiv x \bmod \prod_{j=0}^{\nu} p_{i,j}$, i. e., $H - x \equiv 0 \bmod \prod_{j=0}^{\nu} p_{i,j}$, which is true by definition of $\alpha_j = +1$. Otherwise, $\alpha_j = -1$. Then, $-H \equiv x \bmod \prod_{j=0}^{\nu} p_{i,j}$, i. e., $H + x \equiv 0 \bmod \prod_{j=0}^{\nu} p_{i,j}$, which is again true by the definition of α_j. Thus, the initial statement is correct.

So, as $\alpha_j \in \{-1,+1\}$ for all $j \in \{0,1,\dots,\nu\}$, it holds that $-H \leq x \leq H$. Since the same holds for x', it follows that $|x - x'| \leq 2H$. Let us bound the value of $\lambda_j = \theta_j / (\prod_{i=0, i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k})$. It holds that $\theta_j \leq 2^4 \cdot p^* \prod_{i=1}^{m'} p_i \cdot \prod_{i=0, i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k}$, as $2^3 \cdot p^* \prod_{i=1}^{m'} p_i$ and $\prod_{i=0, i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k}$ are coprime and

thus, the least θ_j satisfying the equivalence conditions in the definition of θ_j is at most their product [149]. The additional factor of 2 is introduced by the inequality constraint $\theta_j \not\equiv 0 \bmod p_{j,1}$, as if the calculated θ_j for the equality constraints does not satisfy that condition, we can extend it to $\theta'_j = \theta_j + 2^3 \cdot p^* \prod_{i=1}^{m'} p_i \cdot \prod_{i=0, i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k}$. This doubles the size estimation and as $p_{j,1}$ is coprime to $2^3 \cdot p^* \prod_{i=1}^{m'} p_i \cdot \prod_{i=0, i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k}$, it holds that θ'_j is not equivalent to 0 mod $p_{j,1}$. Thus,

$$\begin{aligned}
\lambda_j &= \frac{\theta_j}{\prod_{i=0,i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k}} \\
&\leq \frac{2^4 \cdot p^* \prod_{i=1}^{m'} p_i \cdot \prod_{i=0,i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k}}{\prod_{i=0,i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k}} \\
&= \frac{2^4 \cdot p^* \prod_{i=1}^{m'} p_i \cdot K / (\prod_{k=0}^{\nu} p_{j,k})}{\prod_{i=0,i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k}} \\
&= \frac{2^4 \cdot p^* \prod_{i=1}^{m'} p_i \cdot K}{\prod_{i=0}^{\nu} \prod_{k=0}^{\nu} p_{i,k}} \\
&\leq \frac{2^4 \cdot p^* \prod_{i=1}^{m'} p_i \cdot K}{4(\nu+1)2^3 \cdot p^* \prod_{i=1}^{m'} p_i} \\
&= \frac{K}{2(\nu+1)}.
\end{aligned}$$

To validate the fourth estimation, we have to prove that $\prod_{i=0}^{\nu} \prod_{k=0}^{\nu} p_{i,k} \geq 4(\nu+1)2^3 \cdot p^* \prod_{i=1}^{m'} p_i$. As previously shown, it holds that $p^* \leq \prod_{i=m'+1}^{\nu^2+2\nu+1} q_i$. Thus,

$$\begin{aligned}
4(\nu+1)2^3 \cdot p^* \prod_{i=1}^{m'} p_i &\leq 4(\nu+1)2^3 \prod_{i=m'+1}^{\nu^2+2\nu+1} q_i \prod_{i=1}^{m'} p_i \\
&= 4(\nu+1)2^3 \prod_{i=1}^{\nu^2+2\nu+1} q_i.
\end{aligned}$$

Hence, it remains to prove that $\prod_{i=0}^{\nu} \prod_{k=0}^{\nu} p_{i,k} \geq 4(\nu+1)2^3 \prod_{i=1}^{\nu^2+2\nu+1} q_i$. Per definition, $p_{0,0} > \max\{p_{2m'}, q_{11}\}$. Thus, comparing the factors of both products, we see that $4(\nu+1)2^3 \prod_{i=1}^{\nu^2+2\nu+1} q_i$ has $4(\nu+1)2^3$ and the first $\max\{2m', 11\}$ prime numbers smaller than $p_{0,0}$ uniquely. In turn, the product $\prod_{i=0}^{\nu} \prod_{k=0}^{\nu} p_{i,k}$ has the largest $\max\{2m', 11\}$ prime factors uniquely. Let us consider the smallest case where $\max\{2m', 11\} = 4$ as the least, reasonable value for $m' = 2$ (a formula with just one clause is trivial). The smallest, reasonable value for $\nu = 7$ if $m' = 2$ and $n' = 3$ (less than 3 variables is not possible). Now it is easy to prove via manual calculation that the product of $4(\nu+1)2^3$ times the first 4 prime numbers (smaller than $p_{0,0}$) is indeed smaller than the product of the next 4 prime numbers larger than $q_{\nu^2+2\nu+1} = q_{65}$. If m' grows, we get the same number of additional prime factors for both products, whereas each new prime number in $\prod_{i=0}^{\nu} \prod_{k=0}^{\nu} p_{i,k}$ is larger than the additional ones in the other product. If we have larger values

for ν, it only influences the product $4(\nu+1)2^3 \cdot p^* \prod_{i=1}^{m'} p_i$ linearly, whereas for the other product, we start with greater prime numbers, thus having a larger impact on the product. Hence, the estimation is correct for all values.

This term bounds each summand of H, as it considers their largest value to satisfy the constraints as well as the modulo when calculating the values (see definition of θ_k). It follows that $2H = 2\sum_{j=0}^{\nu} \theta_j < 2 \cdot (\nu+1) \cdot K/(2(\nu+1)) = K$. Thus, $x = x'$ and we conclude that solutions of the form $x = \sum_{j=0}^{\nu} \theta_j \alpha_j$ are the unique solutions to the system (P4.1) and (P4.2).

Hence, we can rewrite

$$\sum_{j=0}^{\nu} \theta_j \alpha_j \equiv \tau \bmod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i,\ \alpha_j \in \{-1,+1\}$$

using the system (P4.1) and (P4.2) to

$$0 \leq |x| \leq H,\ x \in \mathbb{Z} \tag{P5.1}$$

$$x \equiv \tau \bmod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i \tag{P5.2}$$

$$(H+x)(H-x) \equiv 0 \bmod K \tag{P5.3}$$

proving their equivalence. □

Next, we rewrite the system (P5) to

$$0 \leq |x| \leq H,\ x \in \mathbb{Z} \tag{P6.1}$$

$$(\tau - x)(\tau + x) \equiv 0 \bmod 2^4 \cdot p^* \prod_{i=1}^{m'} p_i \tag{P6.2}$$

$$(H+x)(H-x) \equiv 0 \bmod K. \tag{P6.3}$$

Claim 22. *The Problem (P5) is a yes-instance if and only if the Problem (P6) is a yes-instance.*

Proof. As only the second conditions differ, we focus on their equivalence in the following. First, we prove that if (P5.2) holds, i. e., $x \equiv \tau \bmod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i$, then (P6.2) holds, i. e., $(\tau - x)(\tau + x) \equiv 0 \bmod 2^4 \cdot p^* \prod_{i=1}^{m'} p_i$. We can rewrite (P5.2) to $x = \lambda 2^3 \cdot p^* \prod_{i=1}^{m'} p_i + \tau$ for some $\lambda \in \mathbb{Z}$. Inserting this in (P6.2) yields:

$$\begin{aligned}&(\tau + \lambda 2^3 \cdot p^* \prod_{i=1}^{m'} p_i + \tau)(\tau - \lambda 2^3 \cdot p^* \prod_{i=1}^{m'} p_i - \tau)\\ &= (2\tau + \lambda 2^3 \cdot p^* \prod_{i=1}^{m'} p_i)(\lambda 2^3 \cdot p^* \prod_{i=1}^{m'} p_i) \equiv 0 \bmod 2^3 \cdot p^* \prod_{i=1}^{m'} p_i\end{aligned}$$

as each factor is multiplied with $\lambda 2^3 \cdot p^* \prod_{i=1}^{m'} p_i$.

Next, we prove the opposite direction. First, observe that if $(\tau - x)(\tau + x) \equiv 0 \bmod 2^4 \cdot p^* \prod_{i=1}^{m'} p_i$, then either $(\tau - x) \equiv 0 \bmod 2^3$ or $(\tau + x) \equiv 0 \bmod 2^3$:

As (P5.2) holds, $(\tau + x) = \lambda_i \cdot 2^i$ and $(\tau - x) = \lambda_j \cdot 2^j$ for some $i, j \in \mathbb{Z}$ and $\lambda_i, \lambda_j \not\equiv 0 \bmod 2$. It follows that

$$\begin{aligned}
&(\tau + x) + (\tau - x) = \lambda_i \cdot 2^i + \lambda_j \cdot 2^j \\
&\Leftrightarrow 2\tau = \lambda_i \cdot 2^i + \lambda_j \cdot 2^j \\
&\Leftrightarrow \tau = \lambda_i \cdot 2^{i-1} + \lambda_j \cdot 2^{j-1}.
\end{aligned}$$

As τ is odd per definition, either i or j has to be 1 and thus, the other parameter has to be 3. Using this, we know that if x satisfies $(P5.2)$, then $(\tau - x) \equiv 0 \bmod 2^3$ or $(\tau + x) \equiv 0 \bmod 2^3$. In the first case, x directly corresponds to a solution of (P6.2), as $x - \tau$ is a multiple of 2^3 and thus, x is a multiple of 2^3 with a residue of τ. Otherwise, $-x$ satisfies the condition using the same argument. Obviously, the other conditions are also satisfied in both systems. □

Lastly, we rewrite the system one final time to

$$0 \leq x \leq H,\ x \in \mathbb{Z} \tag{QC.1}$$

$$\begin{aligned}
&2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i (H^2 - x^2) + K(\tau^2 - x^2) \\
&\equiv 0 \bmod 2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i \cdot K.
\end{aligned} \tag{QC.2}$$

Claim 23. *The Problem (P6) is a yes-instance if and only if the* Quadratic Congruences *problem is a yes-instance.*

Proof. First, as we only consider x^2, we can suppose $x \geq 0$ and thus re-writing (P6.1) to (QC.1) is correct. Further, (P6.2) and (P6.3) merge into (QC.2). Recall that $2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i$ and K are co-prime. The first summand obviously always contains the factor $2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i$, thus we have to find an x such that $(H^2 - x^2) \equiv 0 \bmod K$ which corresponds to (P6.3). The second summand clearly is a multiple of K, thus we have to ensure that $(\tau^2 - x^2) \equiv 0 \bmod 2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i$. This matches (P5.2).

Dissolving the brackets and rearranging the term (QC.2), we get

$$\begin{aligned}
&(2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i + K) x^2 \\
&\equiv K\tau^2 + 2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i H^2 \bmod 2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i \cdot K.
\end{aligned}$$

As $2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i + K$ is relatively prime to $2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i \cdot K$, it has an inverse modulo $2^4 \cdot p^* \cdot \prod_{i=1}^{m'} p_i \cdot K$ [122]. Thus, by multiplying by the inverse, we get the values for α, β and γ as in the transformation above. □

Overall, this proves that satisfying the formula Φ is equivalent to an instance of the Quadratic Congruences problem admitting a feasible solution.

Running time: All steps, numbers and their computation can be bounded in a polynomial dependent of n_3, i. e., the number of variables in the 3-SAT formula, and m_3, i. e., the number of clauses in the formula. First, we eliminate unnecessary clauses from the formula. Thus, we have to go through all clauses once. The first $2m'+1$ prime numbers have a value of at most $O(m'\log(m'))$ and can thus be found in polynomial time via sieving. The function $(4(\nu+1)2^3\prod_{i=1}^{\nu^2+2\nu+1} q_i)^{1/((\nu^2+2\nu+1)\log(\nu^2+2\nu+1))}$ is at most 32 as shown before. Thus, we can also bound the value of the next $\nu^2+2\nu+1$ prime numbers larger than 32 and $p_{2m'}$ by a polynomial in n_3 and m_3 and we can compute them efficiently by sieving. All other numbers calculated in the transformation are a product or sum over these prime numbers (each occurring at most once in the calculation) and thus, their values are also in $\text{poly}(n_3, m_3)$. We can compute the inverse $(2^4 \cdot p^* \prod_{i=1}^{m'} p_i + K)^{-1}$ in polynomial time [122]. □

Now, we have proved that the Quadratic Congruences problem is NP-hard even in the restricted case when all prime factors in β only appear at most once (except 2). To apply the ETH, however, we also have to estimate the dimensions of the generated instance. Denote by $B = b_1^{\beta_1}, \ldots, b_{n_{QC}}^{\beta_{n_{QC}}}$ the prime factorization of β where $b_1, \ldots, b_{n_{QC}}$ denotes the different prime factors of β and β_i the occurrence of b_i. The above reduction yields the following parameters:

Theorem 24. *An instance of the 3-SAT problem with n_3 variables and m_3 clauses is reducible to an instance of the Quadratic Congruences problem in polynomial time with the properties that $\alpha, \beta, \gamma \in 2^{O((n_3+m_3)^2\log(n_3+m_3))}$, $n_{QC} \in O((n_3+m_3)^2)$, $max_i\{b_i\} \in O((n_3+m_3)^2\log(n_3+m_3))$, and each prime factor in β occurs at most once except the prime factor 2 which occurs four times.*

Proof. In Theorem 18, we already showed and proved a reduction from the 3-SAT problem to the Quadratic Congruences problem and argued the running time. It remains to bound the parameters. To do so, we bound the numbers occurring in the reduction above in order of their appearance.

After eliminating the trivial clauses, it obviously holds that $m' \leq m_3$ and $n' \leq n_3$. Next, we calculate $\tau_{\Phi'}$. Its absolute value can be bounded as

$$|\tau_{\Phi'}| = |-\sum_{i=1}^{m'}\prod_{j=1}^{i} p_j| = \sum_{i=1}^{m'}\prod_{j=1}^{i} p_j$$
$$\leq m_3 \prod_{j=1}^{m_3} p_j \leq m_3 2^{2m_3\log(m_3)} \leq 2^{O(m_3\log(m_3))}$$

since the product of the first k prime numbers is bounded by $2^{2k\log(k)}$ for all $k \geq 2$, see Lemma 17. Similarly, $\max_i\{|f_i^+|, |f_i^-|\} \leq \sum_{x_i\in\sigma_j}\prod_{k=1}^{j} p_k + \sum_{\bar{x}_i\in\sigma_j}\prod_{k=1}^{j} p_k \leq 2m_3 \cdot 2^{2m_3\log(m_3)} \leq 2^{O(m_3\log(m_3))}$ and further $\max_j\{c_j\} = \max_j\{\prod_{i=1}^{j} p_i, f_j^+ + f_j^-\} \leq 2^{O(m_3\log(m_3))}$. Per definition, $\nu = 2m' + n' =$

$O(n_3+m_3)$. The largest prime number $max_i\{b_i\}$ we generate in the reduction is p^*, which is the $(\nu^2+2\nu+2m'+13)$th prime number. Thus, its value is bounded by $p^* \leq O(\nu^2\log(\nu)) = O((n_3+m_3)^2\log(n_3+m_3))$ [81]. Due to the modulo, we can bound $\max_j\{\theta_j\}$ as

$$\begin{aligned}\max_j\{\theta_j\} &\leq 2^4 \cdot p^* \prod_{i=1}^{m'} p_i \cdot \prod_{i=0, i\neq j}^{\nu} \prod_{k=0}^{\nu} p_{i,k} \\ &\leq 2^4 2^{O((n_3+m_3)^2\log(n_3+m_3))} = 2^{O((n_3+m_3)^2\log(n_3+m_3))}.\end{aligned}$$

Thus, $H = \sum_{j=0}^{\nu} \theta_j \leq \nu \cdot 2^{O((n_3+m_3)^2\log(n_3+m_3))} = 2^{O((n_3+m_3)^2\log(n_3+m_3))}$ and $K = \prod_{i=0}^{\nu}\prod_{k=0}^{\nu} p_{i,k} \leq 2^{O((n_3+m_3)^2\log(n_3+m_3))}$. Finally, we can bound the main parameters. As α is bounded by the modulo of β, it follows that $\alpha \leq \beta$. Further, $\beta = 2^4 \cdot p^* \prod_{i=1}^{m'} p_i \cdot K \leq 2^{O((n_3+m_3)^2\log(n_3+m_3))}$. Per definition, $\gamma = H$ and thus, $\gamma \leq 2^{O((n_3+m_3)^2\log(n_3+m_3))}$, which finalizes the estimation of the numbers. □

4.2 REDUCTION FROM QUADRATIC CONGRUENCES

This sections presents the reduction from the QUADRATIC CONGRUENCES problem to the 2-STAGE ILP problem. First, we present a transformation of an instance of the QUADRATIC CONGRUENCES problem to an instance of the NON-UNIQUE REMAINDER problem. This problem was not considered so far and serves as an intermediate step in this chapter. However, it might be of independent interest, as it generalizes the prominent Chinese Remainder theorem. Secondly, we show how an instance of the NON-UNIQUE REMAINDER problem can be modeled as a 2-stage stochastic ILP. Recall that in the NON-UNIQUE REMAINDER problem, we are given numbers $x_1,\dots,x_{n_{\text{NR}}}, y_1,\dots,y_{n_{\text{NR}}}, q_1,\dots,q_{n_{\text{NR}}}, \zeta \in \mathbb{N}$ where the q_is are pairwise co-prime. The question is to decide whether there exists a natural number z satisfying the following equations and which is smaller or equal to ζ:

$$\begin{aligned} &z \bmod q_1 \in \{x_1, y_1\} \\ &z \bmod q_2 \in \{x_2, y_2\} \\ &\vdots \\ &z \bmod q_{n_{\text{NR}}} \in \{x_{n_{\text{NR}}}, y_{n_{\text{NR}}}\}. \end{aligned}$$

In other words, we either should meet the residue x_i or y_i. Thus, we can rewrite the equation as $z \equiv x_i \bmod q_i$ or $z \equiv y_i \bmod q_i$ for all i. Indeed, this problem becomes easy if $x_i = y_i$ for all i, i. e., we know the remainder we want to satisfy for each equation [163]: First, compute s_i and r_i with $r_i \cdot q_i + s_i \cdot \prod_{j=1, j\neq i}^{n_{\text{NR}}} q_j = 1$ for all i using the Extended Euclidean algorithm. Now, it holds that $s_i \cdot \prod_{j=1, j\neq i}^{n_{\text{NR}}} q_j \equiv 1 \bmod q_i$, as q_i and $\prod_{j=1, j\neq i}^{n_{\text{NR}}} q_j$ are coprime, and $s_i \cdot \prod_{j=1, j\neq i}^{n_{\text{NR}}} q_j \equiv 0 \bmod q_j$ for $j \neq i$. Thus, the smallest solution corresponds to $z = \sum_{i=1}^{n_{\text{NR}}} x_i \cdot s_i \cdot \prod_{j=1, j\neq i}^{n_{\text{NR}}} q_j$ due to the Chinese Remainder theorem [163]. Comparing z to the bound ζ finally yields the answer. Also note that if

n_{NR} is constant, we can solve the problem by testing all possible vectors $(v_1, \ldots, v_{n_{\text{NR}}})$ with $v_i \in \{x_i, y_i\}$ and then use the Chinese Remainder theorem as explained above.

Theorem 25. *The* Quadratic Congruences *problem is reducible to the* Non-Unique Remainder *problem in polynomial time with the properties that* $n_{NR} \in O(n_{QC})$, $\max_{i\in\{1,\ldots,n_{NR}\}}\{q_i, x_i, y_i\} = O(\max_{j\in\{1,\ldots,n_{QC}\}}\{b_j^{\beta_j}\}$, *and* $\zeta \in O(\gamma)$.

Proof. Transformation: Set $q_1 = b_1^{\beta_1}, \ldots, q_{n_{\text{NR}}} = b_{n_{\text{QC}}}^{\beta_{\text{QC}}}$ and $\zeta = \gamma$ where β_i denotes the occurrence of the prime factor b_i in the prime factorization of β. Compute $\alpha_i \equiv \alpha \bmod q_i$. Set $x_i^2 = \alpha_i$ if there exists such an $x_i \in \mathbb{Z}_{q_i}$. Further, compute $y_i = -x_i + q_i$. If there is no such number x_i and thus y_i, produce a trivial no-instance.

Instance size: The numbers we generate in the reduction equal the prime numbers of the Quadratic Congruences problem including their occurrence. Hence, it holds that $\max_{i\in\{1,\ldots,n_{\text{NR}}\}}\{q_i\} = O(\max_{j\in\{1,\ldots,n_{\text{QC}}\}}\{b_j^{\beta_j}\}$. Due to the modulo, this value also bounds x_i and y_i. The upper bound on a solution equals the ones from the instance of the Quadratic Congruences problem, i. e., $\zeta \in O(\gamma)$, and $n_{\text{NR}} = n_{\text{QC}}$ holds.

Correctness: First, let us verify that producing a trivial no-instance is correct if we cannot find some x_i. Indeed, this can be traced back to the Chinese Remainder theorem: If and only if there is an x with $x^2 \equiv \alpha \bmod \beta$ and $q_1, \ldots, q_{n_{\text{NR}}}$ (i. e., the equivalences to $b_i^{\beta_i}$) is the prime factorization of β, then $x^2 \equiv \alpha_i \bmod q_i$, $\alpha_i \in \mathbb{Z}_{q_i}$ for all i. In other words, it has to be dividable by all $b_i^{\beta_i}$ yielding the same remainder α (modulo $b_i^{\beta_i}$). Hence, if there does not exist a square root of α in one of the systems, then $x^2 \equiv \alpha \bmod \beta$ has no solution.

But if there exist x_i and y_i, these values are in $\mathbb{Z}_{q_i}$, as $x_i \leq \alpha_i < q_i$ per definition of x_i and α_i. Further, both values solve the problem $x_i^2, y_i^2 \equiv \alpha \bmod q_i$ as

$$x_i^2 \equiv \alpha_i \bmod q_i \equiv \alpha_i + \lambda \cdot q_i \bmod q_i \equiv \alpha \bmod q_i$$

for some $\lambda \in \mathbb{N}$. Moreover,

$$\begin{aligned} y_i^2 &\equiv (-x_i + q_i)^2 \bmod q_i = q_i^2 - 2x_i q_i + x_i^2 \bmod q_i \\ &\equiv x_i^2 \bmod q_i \equiv \alpha \bmod q_i. \end{aligned}$$

The third equation holds, as each summand except the last one is a multiple of q_i. The last transformation is true due to the computation above.

Note that for all primes greater than 2 it holds that $x_i \neq y_i$. This can easily be seen, as we already argued that x_i and y_i are in $\mathbb{Z}_{p_i}$. Let us suppose both values are equal, i. e.,

$$\begin{aligned}
&x_i^2 = y_i^2 \\
&\Leftrightarrow \alpha_i = (-x_i + q_i)^2 \\
&\Leftrightarrow \alpha_i = q_i^2 - 2q_i x_i + x_i^2 \\
&\Leftrightarrow \alpha_i = q_i^2 - 2q_i x_i + \alpha_i \\
&\Leftrightarrow 2q_i x_i = q_i^2 \\
&\Leftrightarrow 2x_i = q_i.
\end{aligned}$$

The factor q_i is a product of some prime number greater than 2 by the assumption above. Thus, there is no x_i satisfying the formula.

Let us now prove the equivalence of the reduction.

$\Rightarrow$ Let the instance of the Quadratic Congruences problem be a yes-instance. Then there exists a z satisfying $z^2 \equiv \alpha \bmod \beta$ with $0 < z \leq \gamma$. This solution directly corresponds to a solution of the generated instance of the Non-Unique Remainder problem. First, $z \leq \gamma = \zeta$. Secondly, z satisfies all equations, as it holds that

$$z^2 \equiv \alpha \bmod \beta \equiv \alpha \bmod \prod_{i=1}^{n_{\text{NR}}} b_i^{\beta_i} \equiv \alpha \bmod b_i^{\beta_i} \text{ for all } i.$$

The first equivalence holds, as the $b_i^{\beta_i}$s are the prime factorization of β. The second equivalence is true, as we can decompose the solution as follows: $z^2 = \lambda \cdot \prod_{i=1}^{n} b_i^{\beta_i} + \alpha$ for some $\lambda \in \mathbb{N}$. Thus, the first summand is not only divided without remainder by $\prod_{i=1}^{n_{\text{NR}}} b_i^{\beta_i}$, but also by all primes along with their occurrences alone, leaving only the second summand α as the remainder. Further, since $x_i^2, y_i^2 \equiv \alpha \bmod q_i$ as shown before, it holds that

$$z^2 \equiv \alpha \bmod b_i^{\beta_i} \equiv \alpha \bmod q_i \equiv x_i^2 \equiv y_i^2 \text{ for all } i.$$

Hence, this satisfies all equations of the generated instance of the Non-Unique Remainder problem making it a yes-instance.

$\Leftarrow$ Let the instance of the Non-Unique Remainder problem be a yes-instance. Thus, we could verify that there exists a solution to the given equations smaller than ζ. Let this solution be denoted as z^*. It holds that $z^* \equiv x_i \bmod q_i$ or $z \equiv y_i \bmod q_i$. Let v_i correspond to the residue that was satisfied, i. e., $v_i = x_i$ or $v_i = y_i$. The solution z^* also solves the Quadratic Congruences problem. First, $z^* \leq \zeta = \gamma$. Further, it holds per definition of the numbers that

$$(z^*)^2 \equiv (v_i)^2 \equiv \alpha \bmod q_i \text{ for all } i.$$

As it satisfies all equations simultaneously and the b_i are pairwise co-prime, it follows from the Chinese Remainder theorem that

$$\begin{aligned}&(z^*)^2 \equiv (v_i)^2 \equiv \alpha \bmod q_i \text{ for all } i\\ &\equiv (z^*)^2 \equiv \alpha \bmod \prod_{i=1}^{n_{\text{NR}}} q_i \equiv \alpha \bmod \prod_{i=1}^{n_{\text{QC}}} b_i^{\beta_i} \equiv \alpha \bmod \beta\end{aligned}$$

as the $b_i^{\beta_i}$s are the prime factorization of β.

Running time: Setting the variables accordingly can be done in time polynomial in n_{QC}. Further, computing each x_i, y_i can be done in poly-logarithmic time regarding the largest absolute number for each $i \in \{1, \dots, n_{\text{NR}}\}$ [37]. □

Finally, we reduce the Non-Unique Remainder problem to the 2-stage ILP problem. Note that the considered 2-stage ILP problem is a decision problem. In other words, we only seek to determine whether a feasible solution exists. We neither optimize a solution vector nor are we interested in the solution vector itself.

Theorem 26. *The Non-Unique Remainder problem is reducible to the 2-stage ILP problem in polynomial time with the properties that $n \in O(n_{NR})$, $s, q, t, \|w\|_\infty, \|b\|_\infty, \|\ell\|_\infty \in O(1)$, $\|u\|_\infty \in O(\zeta)$, and $\Delta \in O(\max_i\{q_i\})$.*

Proof. Transformation: Having the instance for the Non-Unique Remainder problem at hand, we construct our ILP as follows with $n = n_{\text{NR}}$:

$$\mathcal{A} \cdot x = \begin{pmatrix} -1 & q_1 & x_1 & y_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ -1 & 0 & \dots & 0 & 0 & \dots & 0 & q_n & x_n & y_n \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 & 1 & 1 \end{pmatrix} \cdot x = b = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

All variables get a lower bound of 0 and an upper bound of ζ. We can set the cost vector arbitrarily, as we are just searching for a feasible solution. Hence, we set it to $w = (0, 0, \dots, 0)^\top$.

Instance size: Due to our construction, it holds that $s = 2, q = 1, t = 3$. The number n of repeated blocks equals the number n_{NR} of equations in the instance of the Non-Unique Remainder problem. The largest entry Δ can be bounded by $\max_i\{q_i\}$. The lower and upper bounds are at most $\|u\|_\infty = O(\zeta)$, $\|\ell\|_\infty = O(1)$. The cost vector w is set to zero and is thus of constant size. The largest value in the right-hand side is $\|b\|_\infty = 1$.

Correctness: $\Rightarrow$ Let the given instance of the Non-Unique Remainder problem be a yes-instance. Thus, there exists a solution $z^* < \zeta$ satisfying all equations. As before, let v_i correspond to the remainder that was satisfied in each equation i, i. e., $v_i = x_i$ or $v_i = y_i$. A solution to our integer linear

program now looks as follows: Set the first variable to z^*. Let the columns corresponding to x_i and y_i be set as follows for each i: If $v_i = x_i$, then set this variable occurrence in the solution vector to 1. Set the occurrence to the corresponding variable of y_i to zero. Otherwise, set the variables the other way round. Finally, the variable corresponding to the columns of the q_i are computed as $(z^* - v_i)/q_i$. It is easy to see that this solution is feasible and satisfies the bounds on the variable sizes.

$\Leftarrow$ Let the given instance of the 2-STAGE ILP problem be a yes-instance. By definition of the constraint matrix we have for every $1 \leq i \leq n$ that there exists a multiple $\lambda_i \geq 0$ such that $z = x_i + \lambda_i q_i$ or $z = y_i + \lambda_i q_i$. Hence, $z \equiv x_i \bmod q_i$ or $z \equiv y_i \bmod q_i$ for every $1 \leq i \leq n$. Further, $z \leq u$. Thus, the solution z is a solution of the NON-UNIQUE REMAINDER problem.

Running time: Mapping the variables and computing the values for the q_is can all be done in polynomial time regarding the largest occurring number and n. □

4.3 RUNTIME BOUNDS FOR 2-STAGE STOCHASTIC ILPS UNDER ETH

This sections presents the proof that the double exponential running time in the current state-of-the-art algorithms is nearly tight assuming the Exponential Time Hypothesis (ETH). To do so, we make use of the reductions above showing that we can transform an instance of the 3-SAT problem to an instance of the 2-STAGE ILP problem.

Corollary 27. *The 2-STAGE ILP problem cannot be solved in time less than $2^{\delta\sqrt{n}}$ for some $\delta > 0$ assuming ETH.*

Proof. Suppose the opposite. That is, there is an algorithm solving the 2-STAGE ILP problem in time less than $2^{\delta\sqrt{n}}$. Let an instance of the 3-SAT problem with n_3 variables and m_3 clauses be given. Due to the Sparsification lemma, we may assume that $m_3 \in O(n_3)$ [92]. The Sparsification lemma states that any 3-SAT formula can be replaced by subexponentially many 3-SAT formulas, each with a linear number of clauses with respect to the number of variables. The original formula is satisfiable if at least one of the new formulas is. This yields that if we cannot decide a 3-SAT problem in subexponential time, we can also not do so for a 3-SAT problem where $m_3 \in O(n_3)$.

We can reduce such an instance to an instance of the QUADRATIC CONGRUENCES problem in polynomial time regarding n_3 such that $n_{\text{QC}} \in O(n_3^2)$, $max_i\{b_i\} \in O(n_3^2 \log(n_3))$, $\alpha, \beta, \gamma = 2^{O(n_3^2 \log(n_3))}$, see Theorems 18 and 24.

Next, we reduce this instance to an instance of the NON-UNIQUE REMAINDER problem. Using Theorem 25, this yields the parameter sizes $n_{\text{NR}} \in O(n_3^2)$, $\max_{i \in \{1,\dots,n_{\text{NR}}\}}\{q_i, x_i, y_i\} = O(n_3^2 \log(n_3))$, and $\zeta \in 2^{O(n_3^2 \log(n_3))}$. Note that all prime numbers greater than 2 appear at most once in the prime factorization of β and 2 appears 4 times. Thus, the largest q_i, which corresponds to

$\max_i\{b_i^{\beta_i}\}$ equals the largest prime number in the QUADRATIC CONGRUENCES problem: The largest prime number is at least the $(\nu^2+2\nu+2m'+13) \geq 13$th prime number by a rough estimation. The 13th prime number is 41 and thus larger than $2^4 = 16$.

Finally, we reduce that instance to an instance of the 2-STAGE ILP problem with parameters $s, q, t, \|w\|_\infty, \|b\|_\infty, \|\ell\|_\infty \in O(1)$, $\|u\|_\infty \in 2^{O(n_3^2 \log(n_3))}$, $n \in O(n_3^2)$, and $\Delta \in O(n_3^2 \log(n_3))$, see Theorem 26.

Hence, if there is an algorithm solving the 2-STAGE ILP problem in time less than $2^{\delta\sqrt{n}}$, this would result in the 3-SAT problem to be solved in time less than $2^{\delta\sqrt{n}} = 2^{\delta\sqrt{C_1 n_3^2}} = 2^{\delta(C_2 n_3))}$ for some constants C_1, C_2. Setting $\delta_3 \leq \delta / C_2$, this would violate the ETH. □

To prove our main result, we still have to reduce the size of the coefficients in the constraint matrix. To do so, we encode large coefficients into submatrices. This reduces the size of the entries greatly while just extending the matrix dimensions slightly. A similar approach was used for example in [114] to prove a lower bound for the size of inclusion minimal kernel-elements of 2-stage stochastic ILPs, or in [118] to decrease the value of Δ in the matrices.

Theorem 28. *The 2-STAGE ILP problem cannot be solved in time less than $2^{2^{\delta(s+t)}}|I|^{O(1)}$ for some constant $\delta > 0$, even if $q = 1$, $\Delta, \|b\|_\infty, \|w\|_\infty \in O(1)$, assuming ETH. Here $|I|$ denotes the encoding length of the total input.*

Proof. First, we show that we can alter the resulting integer linear program such that we reduce the size of Δ to $O(1)$. We do so by encoding large coefficients with base 2, which comes at the cost of enlarged dimensions of the constraint matrix. Let $\text{enc}(x)$ be the encoding of a number x with base 2. Further, let $\text{enc}_i(x)$ be the ith number of $\text{enc}(x)$. Finally, $\text{enc}_0(x)$ denotes the last significant number of the encoding. Hence, the encoding of a number x is $\text{enc}(x) = \text{enc}_0(x)\text{enc}_1(x)\dots\text{enc}_{\lfloor\log(\Delta)\rfloor}(x)$ and x can be reconstructed by $x = \sum_{i=0}^{\lfloor\log(\Delta)\rfloor} \text{enc}_i(x) \cdot 2^i$.

Let a matrix E be defined as,

$$E = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ 0 & 2 & -1 & 0\dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \\ 0 & \dots & 0 & 2 & -1 \end{pmatrix}.$$

We rewrite the constraint matrix as follows: For each coefficient $c > 1$, we insert its encoding $\text{enc}(c)$ and we put the matrix E beneath. Furthermore, we have to fix the dimensions for the first row in the constraint matrix, the columns without great coefficients and the right-hand side b by filling the matrix at the corresponding positions with zeros. In detail, the altered integer linear program $\mathcal{A} \cdot x = b$ looks as follows:

$$\begin{pmatrix}
-1 & \operatorname{enc}(q_1) & \operatorname{enc}(x_1) & \operatorname{enc}(y_1) & 0 & \dots & 0 & 0 & \dots & 0 \\
0 & E & 0\dots0 & 0\dots0 & 0 & \dots & 0 & 0 & \dots & 0 \\
\vdots & 0\dots0 & E & 0\dots0 & 0 & \dots & 0 & 0 & \dots & 0 \\
0 & 0\dots0 & 0\dots0 & E & 0 & \dots & 0 & 0 & \dots & 0 \\
0 & 0\dots0 & 1\,0\dots0 & 1\,0\dots0 & 0 & \dots & 0 & 0 & \dots & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
-1 & 0 & \dots & 0 & 0 & \dots & 0 & \operatorname{enc}(q_n) & \operatorname{enc}(x_n) & \operatorname{enc}(y_n) \\
0 & 0 & \dots & 0 & 0 & \dots & 0 & E & 0\dots0 & 0\dots0 \\
\vdots & 0 & \dots & 0 & 0 & \dots & 0 & 0\dots0 & E & 0\dots0 \\
0 & 0 & \dots & 0 & 0 & \dots & 0 & 0\dots0 & 0\dots0 & E \\
0 & 0 & \dots & 0 & 0 & \dots & 0 & 0\dots0 & 1\,0\dots0 & 1\,0\dots0
\end{pmatrix} \cdot x = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

Note that the ones beneath the sub-matrices $\operatorname{enc}(x_i)$ and $\operatorname{enc}(y_i)$ correspond to $\operatorname{enc}_0(x_i)$ and $\operatorname{enc}_0(y_i)$. The independent blocks consisting of $\operatorname{enc}(c)$ and the matrix E beneath correctly encodes the number $c > 1$, i. e., it preserves the solution space: Let x_c be the number in the solution corresponding to the column with entry c of the original instance. The solution for the altered column (i. e., the sub-matrix) is $(x_c \cdot 2^0, x_c \cdot 2^1, \dots, x_c \cdot 2^{\lfloor \log(\Delta) \rfloor})$. The additional factor of 2 for each subsequent entry is due to the diagonal of E. It is easy to see that $c \cdot x_c = \sum_{i=0}^{\lfloor \log(\Delta) \rfloor} \operatorname{enc}_i(c) \cdot x_c \cdot 2^i$, as we can extract x_c on the right-hand side and solely the encoding of c remains. Thus, the solutions of the original matrix and the altered one directly transfer to each other. Hence, the solution space is preserved.

Regarding the dimensions, each $c > 1$ is replaced by some $O(\log(\Delta) \times O(\log(\Delta))$ matrix. Thus, the dimension expands to $s' = s \cdot O(\log(\Delta)) = O(\log(\Delta))$, $t' = t \cdot O(\log(\Delta)) = O(\log(\Delta))$, while q and n stay the same. Further, we have to adjust the bounds. The lower bound for all new variables is also zero. For the upper bounds, we allow an additional factor of 2^i for the ith value of the encoding. Thus, $\|u'\|_\infty = 2^{\lfloor \log(\Delta) \rfloor} \|u\|_\infty$. Further, we get that the largest coefficient is bounded by $\Delta' = O(1)$. The right-hand side b enlarges to a vector b' with $O(n \log(\Delta))$ entries.

Now suppose there is an algorithm solving the 2-STAGE ILP problem in time less than $2^{2^{\delta(s+t)}} |I|^{O(1)}$. The proof of Theorem 27 shows that we can transform an instance of the 3-SAT problem with n_3 variables and m_3 clauses to an 2-stage stochastic ILP with parameters $s, q, t, \|w\|_\infty, \|b\|_\infty, \|\ell\|_\infty \in O(1)$, $\|u\|_\infty \in 2^{O(n_3^2 \log(n_3))}$, $n \in O(n_3^2)$, and $\Delta \in O(n_3^2 \log(n_3))$. Further, we explained above that we can transform this ILP to an equivalent one where

$$\begin{aligned}
&s' = O(\log(\Delta)) = O(\log(n_3^2 \log(n_3))) = O(\log(n_3)), \\
&t' = O(\log(\Delta)) = O(\log(n_3^2 \log(n_3))) = O(\log(n_3)), \\
&\Delta' = O(1), \\
&b' \in \mathbb{Z}^{O(n_3^2 \log(n_3))}, \\
&\|u'\|_\infty = 2^{\lfloor \log(\Delta) \rfloor} \|u\|_\infty = 2^{\lfloor \log(n_3^2 \log(n_3)) \rfloor} 2^{O(n_3^2 \log(n_3))} \\
&\quad = 2^{O(n_3^2 \log(n_3))},
\end{aligned}$$

while q, and n stay the same. The encoding length $|I|$ is then given by

$$\begin{aligned}|I| = (ns'(q+nt'))\log(\Delta') + (q+nt')\log(\|\ell\|_\infty) + \\ (q+nt')\log(\|u'\|_\infty) + ns'\log(\|b'\|_\infty) + (q+nt')\log(\|w\|_\infty) \\ = 2^{O(n_3^2)}.\end{aligned}$$

Hence, if there is an algorithm solving the 2-STAGE ILP problem in time less than $2^{2^{\delta(s+t)}}|I|^{O(1)}$, this would result in the 3-SAT problem to be solved in time less than

$$\begin{aligned}2^{2^{\delta(s+t)}}|I|^{O(1)} = 2^{2^{\delta(C_1\log(n_3)+C_2\log(n_3))}}2^{n_3^{O(1)}} = 2^{2^{\delta C_3\log(n_3)}}2^{n_3^{O(1)}} \\ = 2^{n_3^{\delta\cdot C_3}}2^{n_3^{O(1)}} = 2^{n_3^{\delta\cdot C_4}}\end{aligned}$$

for some constants C_1, C_2, C_3, C_4. Setting $\delta = \delta'/C_4$, we get $2^{n_3^{\delta C_4}} = 2^{n_3^{\delta'}}$. As it holds for sufficient large x and $\epsilon < 1$ that $x^\epsilon < \epsilon x$, it follows that $2^{n_3^{\delta'}} < 2^{\delta' n_3}$. This violates the ETH. Note that this result even holds if $s = 1$, $\Delta, \|b\|_\infty, \|w\|_\infty \in O(1)$ as constructed by our reductions. □

An author owes a duty to the truth.

— from *The City of Dreaming Books* by Walter Moers

5 TIGHTNESS OF SENSITIVITY AND PROXIMITY BOUNDS

The previous chapters discuss block-structured integer linear programs. The specific forms of their constraint matrices induce certain properties helpful for designing efficient algorithms. In the following, we are interested in some structural properties of general ILPs, i. e., where the constraint matrices have arbitrary integral values in each position. Namely, we focus on two structural measures called *proximity* and *sensitivity*, which arise frequently in the design of approximation schemes and online algorithms (see e. g. [56, 58, 87, 95, 103, 147, 156, 158]).

Consider some general ILP with constraint matrix $A \in \mathbb{Z}^{d_1 \times d_2}$, a right-hand side $b \in \mathbb{Z}^{d_1}$ and a cost vector $w \in \mathbb{Z}^{d_2}$. Let Δ be the largest absolute value of the entries in A. Finally, denote by $\mathrm{subDet}(A)$ the largest determinant of any $d_3 \times d_3$ submatrix of A. In the following, we recall the definitions of the two aforementioned measures. To do so, we heavily reuse the preliminaries introduces in Chapter 2.

The sensitivity $\mathrm{sens}(A, b, b', w)$ of an ILP is defined as the distance between two optimal, integral solutions if the right-hand side changes from b to b', i. e., $\mathrm{sens}(A, b, b', w) = \mathrm{dist}(\mathrm{Sol.int}(A, b, w), \mathrm{Sol.int}(A, b', w))$. A small sensitivity is useful when the problem formulation alters the right-hand side, as it implies that an optimal solution for the new problem is close. Thus, we do not have to change our current, optimal, integral solution x too much. Hence, we can search for it exhaustively or by a dynamic program. Typical applications are online algorithms where items arrive or leave (thus changing the right-hand side corresponding to the current item multiplicities).

The proximity $\mathrm{prox}(A, b, w)$ of an ILP is defined as $\mathrm{dist}(\mathrm{Sol.frac}(A, b, w), \mathrm{Sol.int}(A, b, w))$, i. e., the maximal distance between any optimal, fractional solution and its nearest optimal, integral one. If the proximity is small, i. e., there exists an optimal, integer solution close to any optimal, fractional solution, this allows us to solve the integer linear program efficiently: First, we compute the optimal, fractional solution z. Then, we search for an optimal, integral solution x in the small box implied by the proximity bound around z.

Cook et al. introduced in [35] upper bounds for these values. We presented these results earlier in Chapter 2. For convenience, though, we re-state the propositions.

Proposition 2 (Theorem 5 in [35]). *If* $\mathrm{Sol.int}(A, b, w)$ *and* $\mathrm{Sol.int}(A, b', w)$ *are non-empty, we have* $\mathrm{dist}(x, \mathrm{Sol.int}(A, b', w)) \leq (\|b - b'\|_\infty + 2) \cdot d_2 \cdot \mathrm{subDet}(A)$ *for each* $x \in \mathrm{Sol.int}(A, b, w)$.

Note that this implies that $\text{sens}(A,b,b',w) \leq (\|b-b'\|_\infty+2)\cdot d_2\cdot\text{subDet}(A)$.

Proposition 3 (Theorem 1 in [35]). *If* $\text{Sol.int}(A,b,w)$ *is non-empty, then we have* $\text{dist}(x,\text{Sol.frac}(A,b,w)) \leq d_2\cdot\text{subDet}(A)$ *for each* $x \in \text{Sol.int}(A,b,w)$ *and further, we have* $\text{dist}(z,\text{Sol.int}(A,b,w)) \leq d_2 \cdot \text{subDet}(A)$ *for each* $z \in \text{Sol.frac}(A,b,w)$.

Note that this implies that $\text{prox}(A,b,w) \leq d_2 \cdot \text{subDet}(A)$.

The value for $\text{subDet}(A)$ can be estimated only in dependence on the dimensions of A and Δ for example by using the Hadamard inequality. The Hadamard inequality states that the determinant of a $d_3 \times d_3$ matrix with columns v_i is at most $\prod_{i=1}^{d_3}\|v_i\|$ [79]. Thus, these bounds neither depend on the cost vector w nor on the size of b (only the sensitivity depends on the distance between b and b'). Hence, we drop the cost vector from our notation and write $\text{prox}(A,b)$ (respectively $\text{sens}(A,b,b')$) to reflect this.

While it is known that these bounds are tight, all known examples either have a very small value of $\Delta = 1$, use negative entries in the constraint matrix, or have a non-integral right-hand side [148]. Hence, these lower bounds often do not correspond to instances from algorithmic problems. Nevertheless, knowing the exact bounds is often helpful. For example, the exponent denoted by $C(A_\delta)$ in the running time of the algorithm in [103] is just an upper bound on the proximity of the underlying configuration integer program (IP). Hence, improving this upper bound would directly lead to a better running time.

Concerning the sensitivity, we find such examples in the field of online algorithms. Often times, the requirement that decisions are not allowed to be rewinded is too strict. Hence, [147] introduced the model of the *migration factor* where a bounded amount of rewinding is allowed. The migration factor in their work and in many others (e. g. [95, 56, 58, 156]) are simply given by the sensitivity of the underlying IPs. Again, any improvement on the general sensitivity bounds would directly improve these migration factors.

Considering such general ILPs of the above type with non-negative constraint matrices, we present for each $\Delta > 0$ and each $d_1 > 0$ examples such that $d_2 \in \Theta(d_1)$ and their proximitiy and sensitivity are at least $\Omega(d_2 \cdot \text{subDet}(A))$. Note that $\Delta > 0$ can be chosen arbitrarily, however, we restrict it to be odd or even dependent on the case. Moreover, the developed ILPs are a special case of integer linear programs where each variable corresponds to an integral point within a polytope $\mathcal{P} \subseteq \mathbb{R}^{d_1}$. We call such ILPs *polytopish* and define them formally as

$$\begin{aligned}&\min w^\top x\\&\sum_{p\in\mathcal{P}\cap\mathbb{Z}^{d_1}} x_p p = b\\&x \in \mathbb{Z}_{\geq 0}^{|\mathcal{P}\cap\mathbb{Z}^{d_1}|}.\end{aligned}$$

Formally, we obtain the following results:

Theorem 29. *For each $\Delta > 0$ and each even $d_1 > 0$, there is a non-negative matrix $A \in \mathbb{Z}_{\geq 0}^{d_1 \times d_1}$, a right-hand side $b \in \mathbb{Z}_{\geq 0}^{d_1}$ and a right-hand side $b' \in \mathbb{Z}_{\geq 0}^{d_1}$ with $\|b - b'\|_1 = 1$ such that $\operatorname{sens}(A, b, b') \geq \Omega(d_1 \cdot \operatorname{subDet}(A))$. Furthermore, the underlying ILP is polytopish.*

Theorem 30. *For each $\Delta \geq 2$ and each odd $d_1 > 0$, there is a non-negative matrix $A \in \mathbb{Z}_{\geq 0}^{15d_1 \times 15d_1+6}$ and a right-hand side $b \in \mathbb{Z}_{\geq 0}^{15d_1}$ such that $\operatorname{prox}(A, b) \geq \Omega(d_1 \cdot \operatorname{subDet}(A))$. Furthermore, the underlying ILP is polytopish.*

Such polytopish ILPs often arise in the context of algorithmic applications. Probably the most famous one among such ILPs is the *configuration ILP* introduced by Gilmore and Gomory [69]. It has its origin in the Bin Packing problem and is now used for many packing and scheduling problems (e. g. [6, 70, 98, 97]). Let us recall the Bin Packing problem and define its configuration ILP.

In the Bin Packing problem, we are given N items with sizes $s_1, \ldots, s_N \in (0, 1]$. Classically, the objective is to pack these items into as few unit-sized bins as possible. As some sizes may be equal, i. e., $\{s_1, \ldots, s_N\} = \{s_1, \ldots, s_d\}$ for some $d \leq N$, we can rewrite the instance as a multiplicity vector of sizes, i. e., $b = (a_1, \ldots, a_d)$ where the ith item size occurs a_i times. Further, we define configurations. A configuration $\kappa = (\kappa_1, \ldots, \kappa_d) \in \mathbb{Z}_{\geq 0}^d$ is a multiplicity vector of item sizes such that the sum of their sizes is at most 1, i. e., $\kappa \cdot (s_1, \ldots, s_d)^T \leq 1$, hence a feasible packing for a bin. Define the constraint matrix as the set of feasible configurations (with one configuration per column). We now aim to find a set of configurations such that we cover each item, i. e., the multiplicities of the item sizes of the chosen configurations equal the occurrences of the item sizes from the input. For the above objective, we do so while minimizing the number of used configurations (including how often they are chosen). Note that all fractional solutions to the constraint $\kappa \cdot (s_1, \ldots, s_d)^\top \leq 1$ describe a knapsack polytope $\mathcal{P} := \mathcal{P}_{s_1, \ldots, s_d}$. Hence, the integer linear program (i. e., the *configuration ILP*) can be written as

$$\begin{aligned} &\min \|x\|_1 \\ &\sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} x_p p = b \\ &x \in \mathbb{Z}_{\geq 0}^{|\mathcal{P} \cup \mathbb{Z}^d|}. \end{aligned}$$

To construct the desired lower bounds for the Bin Packing problem, we define item sizes such that the columns C_1 of our examples for the general ILPs are subsets of the columns of the configuration ILPs. Then, we define a new objective where the values of the cost vectors corresponding to the configurations $\kappa \in C_1$ get value 0 and the remaining ones value 1. To minimize this function, we thus cannot take other columns than the ones in C_1. Setting the right-hand side as for the general ILPs essentially yields the same examples. Thus, the same bounds are derived. This construction shows that in order to improve the bounds on the proximity or sensitivity of the Bin Packing problem, the objective function $\min \|x\|_1$ needs to be taken into account.

SUMMARY OF RESULTS

- For each $\Delta > 0$ and each even $d_1 > 0$, we construct an ILP with $d_2 \in \Theta(d_1)$ and a non-negative constraint matrix such that the ILP is polytopish and admits a sensitivity of at least $\Omega(\|b - b'\|_1 d_2 \cdot \text{subDet}(A))$.
- There is a cost vector w such that we derive a lower bound of $\Omega(\|b - b'\|_1 d_2 \cdot \text{subDet}(A))$ on the sensitivity of the configuration ILP of the Bin Packing problem.
- For each $\Delta \geq 2$ and each odd $d_1 > 0$, we construct an ILP with $d_2 \in \Theta(d_1)$ and a non-negative constraint matrix such that the ILP is polytopish and admits a proximity of at least $\Omega(d_2 \cdot \text{subDet}(A))$.
- There is a cost vector w such that we derive a lower bound of $\Omega(d_2 \cdot \text{subDet}(A))$ on the proximity of the configuration ILP of the Bin Packing problem.

FURTHER RELATED WORK Already in 1986, Cook et al. proved upper bounds regarding the proximity and sensitivity for general integer linear programs [35], see Proposition 2 and Proposition 3. Still, these classical bounds are state-of-the-art. This raises the question whether these bounds are tight. In this chapter, we answer this affirmatively.

For the case of $d_1 = 1$, Aliev et al. present a tight lower bound for the proximity of $\|x - z\|_\infty \leq \Delta - 1$ [4]. Further settings were studied as separable convex objective functions [88] or mixed integer constraints [141].

Recently, another proximity bound is proven by Eisenbrand and Weismantel [49]. Using the Steinitz lemma, they show that the ℓ_1-distance of an optimal, fractional solution z and its corresponding integral solution x is bounded by $\|x - z\|_1 \leq d_1 \cdot (2d_1\Delta + 1)^{d_1}$. This result also holds when upper bounds for the variables are present. Further, it is improved to $\|x - z\|_1 < 3d_1^2 \log(2\sqrt{d_1} \cdot \Delta^{1/d_1}) \cdot \Delta$ using sparsity [124].

For n-fold ILPs, it holds that if x is a solution to a right-hand side b and the right-hand side changes to b' still admitting a finite, optimal solution x', then $\|x - x'\|_1 \leq \|b - b'\|_1 \cdot O(rs\Delta)^{rs}$ [102]. In turn, it was shown that the proximity is bounded by $\|x - z\|_1 \leq (rs\Delta)^{O(rs)}$ [39]. Note that both bounds are independent of the number of rows and columns of the complete constraint matrix.

STRUCTURE OF THIS CHAPTER Section 5.1 presents the examples to obtain the desired lower bounds on the sensitivity of general ILPs. These examples are then transferred to the Bin Packing problem, deriving the same bounds. Similarly, Section 5.2 proves the lower bounds on general ILPs regarding proximity. Then, these examples are used to obtain the same bounds on the proximity of the Bin Packing problem.

5.1 SENSITIVITY OF ILPS

In this section, we provide lower bounds for the sensitivity of ILPs. First, we present an example for general ILPs. Then, we show how we can use this example to prove the same bound for the ILPs which arise from the Bin Packing polytope.

Sensitivity of General ILPs

This section proves the sensitivity bound for general ILPs, i. e., a lower bound on the distance between $\text{Sol.int}(A,b)$ and $\text{Sol.int}(A,b')$. Let d_1 be an even number and $\Delta \in \mathbb{N}_{>0}$. We consider the following ILP (ILP I) with a cost vector $w = (0,\dots,0)$ (corresponding to no objective function):

$$\underbrace{\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ \Delta & 1 & \cdots & 0 & 0 \\ 0 & \Delta & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & \Delta & 1 \end{pmatrix}}_{=:A} x = \begin{pmatrix} 1 \\ \Delta \\ \Delta^2 \\ \vdots \\ \Delta^{d_1-2} \\ \Delta^{d_1-1} \end{pmatrix}. \tag{ILP I}$$

Note that the ILP (ILP I) is polytopish. To see this, let $A_1,\dots,A_{d_1}$ be the columns of the ILP and define $\mathcal{P} = \text{conv}\{A_1,\dots,A_{d_1}\}$ as the convex hull of the columns. We prove in the following that the integer points in $\mathcal{P}$ are exactly the columns themselves.

Claim 31. *It holds that* $\text{conv}\{A_1,A_2,\dots,A_{d_1}\} \cap \mathbb{Z}^{d_1} = \{A_1,A_2,\dots,A_{d_1}\}$.

Proof. Let $x = (x_1,\dots,x_{d_1})$ be a convex combination of the columns where Ax is an integer point. Let i be the first column with $0 < x_i < 1$. The ith row appears with a non-zero entry only in the columns i and $i-1$. Since x_{i-1} is not fractional per definition, the ith entry of Ax is fractional. This is a contradiction. Hence, there are no fractional values in x and therefore exactly one entry is 1 and all others are 0. □

Hence, the ILP is polytopish. Next, we prove the main result of this section.

Theorem 29. *For each $\Delta > 0$ and each even $d_1 > 0$, there is a non-negative matrix $A \in \mathbb{Z}_{\geq 0}^{d_1 \times d_1}$, a right-hand side $b \in \mathbb{Z}_{\geq 0}^{d_1}$ and a right-hand side $b' \in \mathbb{Z}_{\geq 0}^{d_1}$ with $\|b-b'\|_1 = 1$ such that $\text{sens}(A,b,b') \geq \Omega(d_1 \cdot \text{subDet}(A))$. Furthermore, the underlying ILP is polytopish.*

Proof. The largest determinant of any quadratic submatrix of the above constraint matrix can be bounded by $\text{subDet}(A) = \Delta^{\Theta(d_1)}$ by the Leibniz formula for determinants.

An optimal solution to the ILP above is clearly unique (Note that we set the objective function to zero, thus optimality corresponds to feasibility.).

We only have one column with a non-zero entry for the first row. Thus, the right-hand side b determines this value. By that, we have only one free, non-zero variable for the second row. Using this argument inductively, we get a unique solution of form $x = (1, 0, \Delta^2, 0, \Delta^4, \dots, \Delta^{d_1-2}, 0)$.

If we now change the first entry of the right-hand side to 0, we again get a unique solution for b' due to the same argument as above. The solution is of form: $x' = (0, \Delta, 0, \Delta^3, \dots, \Delta^{d_1-1})$. As the zero and non-zero entries switch in the solution vectors, the difference is the sum of their entries, i. e., $\|x-x'\|_1 \geq \Omega(\|b-b'\|_1 \Delta^{\Theta(d_1)}) = \Omega(d_1 \cdot \text{subDet}(A))$ implying the statement. The ILP is polytopish due to Claim 31. □

Sensitivity of the Bin Packing ILP

Let us now construct an example where the sensitivity for the Bin Packing polytope is large. In this problem, we are given N items with d different sizes. Define these sizes as $s_i = 1/(2\Delta) + i \cdot \epsilon$ for $i = 1, \dots, d$ and some $\epsilon > 0$ with $\epsilon \leq \frac{1}{4(d-1+\Delta d)}$. Obviously, the constraint matrix from the previous example is a subset of feasible configurations, i. e., a subset of the columns of the constraint matrix for this problem, as

$$\begin{aligned} & s_i + \Delta s_{i+1} \leq s_{d-1} + \Delta s_d \\ & = 1/(2\Delta) + (d-1)\epsilon + 1/2 + \Delta d\epsilon \\ & = 1/2 + 1/(2\Delta) + \epsilon(d - 1 + \Delta d) \\ & \underbrace{\leq}_{d \geq 2} 1/2 + 1/4 + \epsilon(d - 1 + \Delta d) \\ & \underbrace{\leq}_{\epsilon \leq \frac{1}{4(d-1+\Delta d)}} 1/2 + 1/4 + 1/4 = 1. \end{aligned}$$

Define by C_1 the set of these columns. Let us now define a linear objective function with a cost vector w which has a 0 entry for each configuration $\kappa \in C_1$ and 1 otherwise. Thus, to minimize the objective function $w^\top x$, we can only choose configurations from C_1. Setting and changing the right-hand side as in the previous example clearly leads to the same sensitivity bound. Combining it with the result of Cook et al. , we thus get:

Corollary 32. *There is a cost vector w such that for the configuration ILP of the Bin Packing problem with constraint matrix A and right-hand sides b and b', we have* $\text{sens}(A, b, b', w) \geq \Omega(\|b - b'\|_1 d_2 \cdot \text{subDet}(A))$.

Hence, if one aims to improve the sensitivity of the configuration ILP of the Bin Packing problem, the special objective function $\|x\|_1$ needs to be taken into account.

5.2 PROXIMITY OF ILPS

This section presents an example for general ILPs where the optimal integer solution x differs greatly from the corresponding fractional solution z, i. e.,

$\|x - z\|_1 \geq \Omega(d_2 \cdot \text{subDet}(A))$. By this, we give a lower bound on the proximity of general ILPs which meets the upper bound for ILPs shown by Cook et al. implying their tightness. Further, we use this example to construct an instance of the Bin Packing problem where the same bound is met.

Proximity of General ILPs

To construct this example, we make use of the Petersen graph. This graph $P = (V, E)$ has fifteen edges, ten vertices and six perfect matchings. A perfect matching E_M is a set of edges such that each vertex $v \in V$ is part of exactly one edge, i. e., there exists exactly one edge $e = (u, w) \in E_M$ satisfying $v = u$ or $v = w$. The Petersen graph has the nice property that every edge is part of exactly two perfect matchings and every two perfect matchings share exactly one edge [2]. The graph and its perfect matchings are displayed in Figure 5.1.

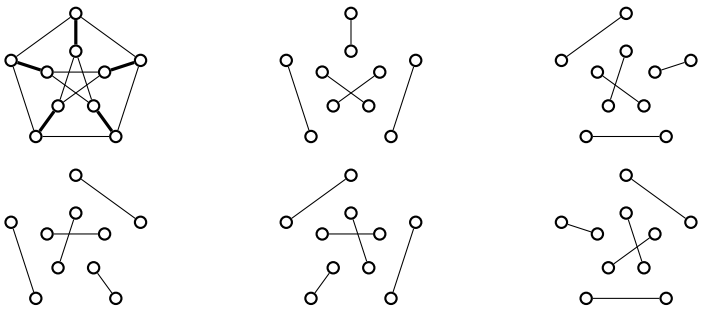

Figure 5.1: The first sub-figure presents the complete Petersen graph with one perfect matching marked by thick edges. The remaining sub-figures each present one of the remaining five perfect matchings.

The Petersen graph is named after its appearance in a paper written by Petersen [144] in 1898. However, it was first mentioned as far back as 1886 [112]. This graph is often used to construct counter-examples for various conjectures due to its neat structure and nice properties. For instance, it was used in [27] to construct small examples where the Round-up Property for Bin Packing instances does not hold. For a survey concerning this graph and more applications, we refer to [90].

We set up a constraint matrix $A_M \in \{0, 1\}^{15 \times 6}$ where each row represents an edge in the Petersen graph and every column corresponds to the indicator vector of one of the perfect matchings. Denote by I the identity matrix of size (15×15). An identity matrix is a matrix where all entries are zero except the diagonal being 1. Further, let $\Delta \geq 2$ and d_1 be an odd number. Construct the ILP (ILP II) as follows, where the cost vector is zero again, i. e., $w = (0, \ldots, 0)$:

$$\underbrace{\begin{pmatrix} A_M & I & 0 & \cdots & 0 & 0 \\ 0 & \Delta \cdot I & I & \cdots & 0 & 0 \\ 0 & 0 & \Delta \cdot I & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & I & 0 \\ 0 & 0 & 0 & \cdots & \Delta \cdot I & I \end{pmatrix}}_{=:A} x = \begin{pmatrix} (1,\dots,1)^T \in \mathbb{N}^{15} \\ (\Delta,\dots,\Delta)^T \in \mathbb{N}^{15} \\ (\Delta^2,\dots,\Delta^2)^T \in \mathbb{N}^{15} \\ \vdots \\ (\Delta^{d_1-1},\dots,\Delta^{d_1-1})^T \in \mathbb{N}^{15} \\ (\Delta^{d_1},\dots,\Delta^{d_1})^T \in \mathbb{N}^{15} \end{pmatrix}.$$

(ILP II)

The first six columns corresponding to the perfect matchings are called *matching columns*. Further, we want a solution where $z \in [0,1]^{6+15\cdot d_1}$ for the fractional case and $x \in \{0,1\}^{6+15\cdot d_1}$ for the integral one. To show that the ILP is polytopish, we argue as before. For readability, set $\nu = 6 + 15 \cdot d_1$. Let $A_1,\dots,A_\nu$ be the columns of A and define $\mathcal{P} = \operatorname{conv}\{A_1,\dots,A_\nu\}$. Next, we argue that the integer points in $\mathcal{P}$ are again only the columns themselves.

Claim 33. *It holds that* $\operatorname{conv}\{A_1,A_2,\dots,A_\nu\} \cap \mathbb{Z}^\nu = \{A_1,A_2,\dots,A_\nu\}$.

Proof. Let $x_1,\dots,x_\nu \in [0,1]$ with $\sum_{i=1}^{\nu} x_i = 1$ such that $A(x_1\dots,x_\nu)$ is integral. First, suppose that $0 < x_i < 1$ for some column $1 \le i \le 6$ corresponding to a matching column. Then, to obtain an integral point $A(x_1,\dots,x_\nu)$, we need to choose another set of columns $C_3 \subseteq \{1,\dots,21\} \setminus \{i\}$ with $x_j > 0$ for all $j \in C_3$. For each such j, there is a row r_j where column j has a value 0 and column i has value 1, as two matchings only share one edge and the identity matrix only has one non-zero entry in each column. Hence, one cannot choose the coefficients x_j for $j \in C_3$ such that $A(x_1,\dots,x_\nu)$ is integral and $\sum_{i=1}^{\nu} x_i = 1$. Thus, the coefficients of the matching columns must be integral.

Now, consider the remaining columns. Suppose there is a column $i > 6$ with $0 < x_i < 1$. If $i \le 21$, this column corresponds to the first identity matrix. As no other column has entries in the first 15 rows, the resulting point $A(x_1,\dots,x_\nu)$ cannot be integral. If $i \le 36$, the only other columns that have non-zero entries have index smaller than 21 and can thus not be fractional as explained before. Using this argument inductively, we see that all solutions for this ILP are integral and thus, the assumption holds. □

Next we estimate the ℓ_1 norm of a (fractional) solution. Define $p = \sum_{i=1}^{(d_1-1)/2} \Delta^{2i-1}$ and $q = \sum_{i=1}^{(d_1-1)/2} \Delta^{2i-2} = \Delta \cdot \sum_{i=0}^{(d_1-1)/2} \Delta^{2i-1} = \Delta \cdot p$.

Claim 34. *The ℓ_1-norm of any (fractional) solution x is at least* $\|x\|_1 = \|y^M\|_1 + \|y^I\|_1 \ge \|y^M\|_1 + (15 - \|y^M\|_1) \cdot \Delta \cdot p + \|y^M\|_1 \cdot p$.

Proof. Consider a (fractional) solution $x = (y^M, y^I)$, where y^M corresponds to the first 6 columns, i. e., to the matching columns. Likewise, divide $A = (A^M, A^I)$ into matching and non-matching columns. The value for the right-hand side given by y^M covers some part of the first 15 rows, namely $0 \le \|A^M y^M\|_1 \le 15$. This can be seen as we can choose at most $\|y^M\|_1 \le 3$

columns (fractionally) such that edges are not overlapping (We have 15 rows in A_M and each column has five 1 entries.). If we would choose more, edges would be overlapping and thus, the right-hand side would be greater than 1 and infeasible. Combining this, we get at most $\|y^M\|_1 \cdot 5 \leq 3 \cdot 5 = 15$ as each perfect matching (and thus column) admits exactly 5 edges.

Let $i \leq 15$ and $b_i := (A^M \cdot y^M)_i$, i. e., the right-hand side covered by the matching columns at position i. We still have to add the value $1 - b_i$ to satisfy the right-hand side. Thus, we set the variables $y^I_i = 1 - b_i$ for $i = 1, \dots, 15$. Further,

$$y^I_{i+1\cdot 15} = \Delta - \Delta(1 - b_i) = \Delta(1 - (1 - b_i)) = \Delta \cdot b_i$$

as these are the only free variable to satisfy the right-hand side, which already has the value $\Delta(1 - b_i)$. In turn, this determines the next 15 variables, i. e.,

$$y^I_{i+2\cdot 15} = \Delta^2 - \Delta^2(b_i) = \Delta^2(1 - b_i).$$

Proceeding with setting the only free variables for the next 15 rows to be satisfied, we get $y^I_{i+3\cdot 15} = \Delta^3 b_i$. The scheme proceeds like this. Therefore,

$$\begin{aligned}
\|y^I\|_1 &= \sum_{i=1}^{15} [\sum_{j=1}^{(d_1+1)/2} (1 - b_i) \cdot \Delta^{2i-2} + \sum_{j=1}^{(d_1-1)/2} b_i \cdot \Delta^{2i-1}] \\
&= \sum_{i=1}^{15} (1 - b_i)(\sum_{j=1}^{(d_1+1)/2} \cdot \Delta^{2i-2}) + \sum_{i=1}^{15} b_i (\sum_{j=1}^{(d_1-1)/2} \cdot \Delta^{2i-1}) \\
&= (15 - \|b\|_1)(\sum_{j=1}^{(d_1+1)/2} \cdot \Delta^{2i-2}) + \|b_i\|_1 (\sum_{j=1}^{(d_1-1)/2} \cdot \Delta^{2i-1}) \\
&= (15 - \|b\|_1)q + \|b_i\|_1 p \geq (15 - \|y^M\|_1) \cdot \Delta \cdot p + \|y^M\|_1 \cdot p.
\end{aligned}$$

Thus, $\|x\|_1 = \|y^M\|_1 + \|y^I\|_1 = \|y^M\|_1 + (15 - \|y^M\|_1) \cdot \Delta \cdot p + \|y^M\|_1 \cdot p$ completing the proof. □

Theorem 30. *For each $\Delta \geq 2$ and each odd $d_1 > 0$, there is a non-negative matrix $A \in \mathbb{Z}_{\geq 0}^{15d_1 \times 15d_1+6}$ and a right-hand side $b \in \mathbb{Z}_{\geq 0}^{15d_1}$ such that $\operatorname{prox}(A, b) \geq \Omega(d_1 \cdot \operatorname{subDet}(A))$. Furthermore, the underlying ILP is polytopish.*

Proof. The largest determinant of any quadratic submatrix of the constraint matrix in (ILP II) can be bounded by $\operatorname{subDet}(A) = \Delta^{\Theta(d_1)}$ using the Leibniz formula for determinants.

One optimal fractional solution z is to take each matching column $1/2$ times, i. e.,

$$z = (\frac{1}{2}, \dots, \frac{1}{2}, 0, \dots 0, \Delta, \dots, \Delta, 0, \dots 0, \Delta^2, \dots, \Delta^2, \dots, \Delta^{d_1-1}, \\ \dots, \Delta^{d_1-1}, 0, \dots 0).$$

Note that again optimality corresponds to feasibility, as we have set the cost vector to zero. It is easy to verify that this solution is feasible, as in

the first 15 columns, we have two entries in M with value 1. Taking both $1/2$ often and setting all variables of the identity matrix to zero gives the right-hand side 1. Then, again the values for the remaining columns are determined as explained in Theorem 29.

In turn, an optimal integral solution would either avoid all matching columns or take some of them. In the first case, this would lead us to take all columns of the first identity matrix and all other values would be determined again and thus, the solution looks like

$$x = (0,\dots 0,1,\dots,1,0,\dots,0,\Delta^2,\dots,\Delta^2,0,\dots 0,\dots,\Delta^{d_1},\dots,\Delta^{d_1}).$$

In the second case, at most one matching column is chosen, as two would already give a too large right-hand side in the first 15 rows (every two matchings share one edge). Then, the identity matrix takes the column corresponding to rows which have a zero entry in the chosen matching column. The remaining solution is then determined by the free variables and the right-hand side as explained in Claim 34.

Now, let us look at the difference of the optimal fractional solution z and the optimal integral ones. It is easy to see that when the optimal solution takes no matching columns, every non-zero component in z is zero in x and vice versa. As the sum of both solutions is $\Delta^{\Theta(d_1)}$, their difference is $\Delta^{\Theta(d_1)}$. For the other case where a matching column is used, we get that the matching columns differ in all positions leading to a difference of all other positions. In particular, few matching columns are used and the remaining ones are zero. By that, we have to take few columns of the identity matrix beside to satisfy the right-hand side. Again, all other entries are determined by that, i. e., we have to take some of the columns of the next identity matrix and so on. Thus,

$$\begin{aligned} \|x-z\|_1 &\geq |\|x\|_1 - \|z\|_1| \geq (1+p+2q) - (15\cdot\Delta\cdot p) \\ &= |1+p+2\Delta p - 15\Delta p| = |(1-13\Delta)p+1| \geq 13\Delta p = \Delta^{\Theta(d_1)}. \end{aligned}$$

Hence, the difference between any (optimal) fractional solution z and an (optimal) integral one x is $\|x-z\|_1 \geq \Delta^{\Theta(d_1)} = \Omega(d_1 \cdot \mathrm{subDet}(A))$. Further, the ILP is polytopish due to Claim 33 completing the proof. □

Proximity of the Bin Packing ILP

We construct an example with a huge proximity by relying on the previous construction. Recall that we are given N items with d different sizes in this problem. Define these sizes as $s_i = 1/(30\Delta) + i\cdot\epsilon$ for $i = 1,\dots,d$ and some $\epsilon > 0$ with $\epsilon \leq \frac{57}{60(d-2+\Delta(d-1))}$. Obviously, the constraint matrix from the previous example is a subset of feasible configurations, i. e., a subset of the

columns of the constraint matrix for this problem, as the value of the largest configuration is bounded by

$$\begin{aligned}
&s_i + \Delta s_{i+1} \leq s_{d-2} + \Delta s_{d-1} \\
&= 1/(30\Delta) + (d-2)\epsilon + 1/30 + \Delta(d-1)\epsilon \\
&= 1/30 + 1/(30\Delta) + \epsilon(d - 2 + \Delta(d-1)) \\
&\underbrace{\leq}_{d \geq 2} 1/30 + 1/60 + \epsilon(d - 2 + \Delta(d-1)) \\
&\underbrace{\leq}_{\epsilon \leq \frac{57}{60(d-2+\Delta(d-1))}} 1/30 + 1/60 + 57/60 = 1.
\end{aligned}$$

Define by C_1 the set of these columns. Let us now define a linear objective function with a cost vector w, which has a 0 entry for each configuration $\kappa \in C_1$ and 1 otherwise. Thus, to minimize the objective function, we can only choose configurations from C_1. Computing a fractional, optimal solution and an optimal, integral one for the right-hand side as of the example above, this clearly leads to the same proximity. Combining it with the result of Cook et. al., we get:

Corollary 35. *There is a cost vector w such that for the configuration ILP of the Bin Packing problem with constraint matrix A and right-hand side b, we have* $\text{prox}(A, b, w) \geq \Omega(d_2 \cdot \text{subDet}(A))$.

Hence, if one aims to improve the proximity of the configuration ILP, the special objective function $\|x\|_1$ needs to be taken into account.

We have a new problem, fresh from the oven and hot as hell.

— from *The Lies of Locke Lamora* by Scott Lynch

6 OPEN QUESTIONS

In this part, various results for the Integer Linear Programming problem were presented. In particular, we discussed the first near-linear time algorithm for n-fold ILPs with respect to the number of variables and a single exponential dependency on $\text{poly}(r,s)$. Further, we proved that the ILPs with a transposed constraint matrix regarding n-fold ILPs, called 2-stage stochastic, are intrinsically harder to solve. We do so by presenting a double exponential lower bound on the block dimensions assuming ETH. For general ILPs, we gave examples where the sensitivity and proximity bounds of Cook et al. are met even if we only have positive, integral entries or if the underlying ILP describes the Bin Packing polytope.

All these results enrich the landscape of the Integer Linear Programming problem. However, there are still a plethora of open questions. In the following, we address those which seem to be the most interesting ones regarding our perspective.

The current state-of-the art algorithms for n-fold ILPs share the trait that we either have a (at least linear) dependency on the encoding length of the largest number in the input, see for example Chapter 3, or the dependency on n is of form $n\log^{f(r,s)}(n)$ for some function f [39]. Thus, it would be interesting if we can find an algorithm which is strongly polynomial and has a linear dependency on n. As an intermediate step, one could study if the exponent of the log-factors in [39] can be reduced to a constant, i. e., if a dependency on n of form $n\log^{O(1)}(n)$ is possible. Since this exponent is introduced while solving the LP relaxation, an equivalent goal for the intermediate step is to find an algorithm with that particular dependency on n for solving the LP.

Regarding the Two-Stage Stochastic Integer Linear Programming problem, the lower bound implies that the current state-of-the-art algorithms are nearly tight. An extension of the 2-stage stochastic ILPs are so-called multi-stage stochastic ILPs where the constraint matrix has a recursive 2-stage stochastic form. In particular, the constraint matrix is a 2-stage stochastic one where the blocks along the diagonal are recursive 2-stage stochastic matrices themselves until in the last recursion depth, we are given an arbitrary block matrix. The current best algorithm has a running time with a tower of exponents of height equal to the recursion depth of the multi-stage stochastic constraint matrix [48]. This raises the question whether the tower of exponents is indeed necessary, or if the complexity collapses at a certain recursion depth.

All the results and questions above consider the block-structured ILPs parameterized over their block dimensions and the largest entry of the constraint matrix. However, there are other natural and closely related parameters one could consider. For example, a parameter which generalizes the block-structured ILPs is the *treedepth* of the graph that results if we have a node for each column and their (non-zero) appearance in the same row is represented by an edge (or, vice versa, if we have a node for each row and an edge represents that two rows share the same variable). Roughly speaking, a higher entanglement of the blocks yields a higher treedepth. For this parameter, various results are known, see for example [48]. However, these results are not tight and thus, studying lower and upper bounds would broaden our understanding about their complexity.

Another natural and intensively studied parameterization is the combination of the number of rows m and the largest absolute value of the constraint matrix Δ. Note that this is closely related to the block-structured ILPs in the case that we only have one block. Thus, lower bounds directly transfer. The best algorithm to solve ILPs parameterized by m and Δ admits a running time of $(\Delta m)^{O(m^2)}n$ if upper bounds on the variables are given [49]. An interesting open question is whether the quadratic dependency in the exponent is necessary or if an algorithm of form $(\Delta m)^{O(m)}n^{O(1)}$ is possible. Note that a running time of form $(\Delta m)^{o(m)}n$ is excluded assuming ETH [118]. Further, in the case of unbounded variables, there exists a tight algorithm with running time $O(\Delta m)^{2m}\log(\|b\|_\infty) + O(mn)$ [104].

Related to the last chapter of this part, it is an interesting question whether the sensitivity and proximity bounds also hold if we consider the classical Bin Packing problem. Thus, in contrast to the examples of Chapter 5, the objective function is the minimization of the number of used bins. If better values for the sensitivity and proximity can be proven, various algorithms for this problem directly improve, see for example [56, 95].

Part II

ALLOCATION PROBLEMS

Beware what you speak, for indeed the words we speak make shadows of what is to come, and by speaking them we bring them to pass, my king.

— from *The Mists of Avalon* by Marion Zimmer Bradley

7

SCHEDULING WITH CLIQUE INCOMPATIBILITIES

In this part, we turn our attention to applications of the aforementioned integer programs. We show that we can solve various variants of the SCHEDULING and BIN PACKING problems by modeling them as (configuration or block-structured) IPs. Applying the state-of-the-art algorithms to solve them or utilizing nice properties such as the sensitivity finally yields the efficient algorithms for these allocation problems. In the following, we present the different variants chapter-wise.

Here, we start with considering non-preemptive scheduling with incompatibilities between jobs. Recall that in the classical SCHEDULING problem, we are given a set $\mathcal{J}$ of N jobs, a set $\mathcal{M}$ of M machines, and a processing time p_j for each job j. On unrelated machines, we have a processing time for each pair of job j and machine i, i. e., we have $p_j^i \in \mathbb{N} \cup \{\infty\}$ stating the processing time of job j if scheduled on machine i. Here, a processing time of $p_j^i = \infty$ means that the job is not allowed to be scheduled on machine i. Further, each job j also admits some weight w_j (in the unweighted case assume all $w_j = 1$). The goal is to find a schedule which minimizes the sum of (weighted) completion times. The completion time C_j of a job j is given by the sum of its starting time and processing time. Thus, we normally also have to state the starting times of the jobs beside the assignment of them onto the machines. However, by the fact that for any machine, we can order the jobs assigned to it optimally by Smith's rule, we do not specify the starting times explicitly. Smith's rule states that it is optimal to schedule the jobs non-increasingly regarding $\phi_i(j) = w_j / p_j^i$ [71].

To model the incompatibilities, assume that the jobs form a graph $G = (\mathcal{J}, E)$ and no two jobs $j, j' \in \mathcal{J}$ connected by an edge $\{j, j'\} \in E$ are allowed to be assigned onto the same machine in a feasible schedule. The graph G is part of the input. In the following, we study the case where the graph is a collection of disjoint cliques. Scheduling with incompatibilities between jobs represents a well-established line of research in scheduling theory and the case of disjoint cliques has received increasing attention in recent years [42, 63, 64, 127].

To understand the importance of this problem setting, consider a task system under difficult conditions like high electromagnetic radiation or with an unstable power supply. Due to the environmental conditions, users prepare tasks in groups and want the jobs in a given group to be scheduled on different processors. That ensures that even if a few processors fail, another processor is able to execute at least parts of the jobs. Further, due to the instability, our

system might even stop working completely and in this case, all jobs that are done only partially have to be scheduled again. As observed in [33] and further pointed out in [26], the sum of completion times criterion tends to reduce this mean number of unfinished jobs at each moment in the schedule.

In the following, we use the three-field notation prevalent in scheduling theory. For instance, makespan minimization on identical machines is abbreviated as $P||C_{\max}$ and minimization of the (weighted) total completion time on unrelated machines as $R||(w_j)\sum C_j$. We denote the here studied problems with clique incompatibility given by a graph as $P|\text{cliques}|\sum(w_j)C_j$ or $R|\text{cliques}|\sum(w_j)C_j$. For a general overview of scheduling notation, we refer the reader to [24].

In this chapter, we study the problem under the paradigm of fixed-parameter tractable algorithms. First, we consider a problem variant with assignment restrictions for the cliques rather than the jobs, i. e., all jobs from the same clique are only allowed to be scheduled on the same set of machines. We denote this problem as $P|\text{cliques}, M(k)|\sum C_j$. We prove that it can be solved in FPT with respect to the number of cliques. Moreover, we show that the problem on unrelated machines can be solved in FPT for the parameter combinations: either largest processing time, number of job kinds and number of machines or largest processing time, number of job kinds, and number of cliques. Here, job *kind* denotes that jobs of the same kind are indistinguishable, i. e., the jobs each have the same processing time vectors and weights. Note that the latter algorithms are a natural extension of the known results due to Knop and Koutecký [115] for the case without incompatibilities.

All FPT results make use of n-fold IPs. In particular, we show that the problems can be expressed as configuration IPs (including carefully extending the ones in [115]) with that specific structure. Further, we also need to introduce appropriate objective functions or extend the existing ones such that they handle the given goal and are separable convex. A function $g : \mathbb{R}^n \to \mathbb{R}$ is called *separable convex* if there exist convex functions $g_i : \mathbb{R} \to \mathbb{R}$ for each $i \in [n]$ such that $g(x) = \sum_{i=1}^{n} g_i(x_i)$. Assume we are given such a function $f : \mathbb{R}^{nt} \to \mathbb{R}$ for our n-fold IP. Denote by $b \in \mathbb{Z}^{r+ns}$ its right-hand side and let ℓ and u be some lower and upper bounds on the variables. Using the result from [39], these n-fold IPs with separable convex objective functions are solvable efficiently:

Proposition 36 ([48]). *Let* $f_{\max} = \max\{|f(x)| \mid \ell_i \le x_i \le u_i \text{ for all } i\}$. *The n-fold IP can be solved in time* $(\Delta rs)^{O(r^2s+rs^2)} nt \log(nt) \log(\|u-\ell\|_\infty) \log(f_{\max})$.

Finally, we can use this solution to construct the desired, optimal schedule.

SUMMARY OF RESULTS

- The problem $P|\text{cliques}, M(k)|\sum C_j$ can be solved in FPT parameterized by the number of cliques c. We show this result by setting up an integer program with n-fold structure and defining an appropriate, separable convex objective function. To do so, we have to argue and incorporate certain structures of optimal solutions.

- The problem $R|\text{cliques}|\sum w_jC_j$ can be solved in FPT parameterized by the number of machines M, the largest processing time $p_{\max}$ and the number of job kinds ϑ. We prove this by developing an appropriate n-fold IP.

- The problem $R|\text{cliques}|\sum w_jC_j$ can be solved in FPT parameterized by the number of cliques c, the largest processing time $p_{\max}$ and the number of job kinds ϑ. Again, the main idea is to model the problem as an n-fold IP.

FURTHER RELATED WORK. Probably the most studied objective function for scheduling is the minimization of the makespan $C_{\max} = \max_j C_j$, directly followed by the minimization of the total completion time objective $\sum C_j$ or respectively the sum of weighted completion times $\sum w_jC_j$. Both problems $P||\sum w_jC_j$ and $P||C_{\max}$ are well-known to be strongly NP-hard. On the other hand, $P||\sum C_j$ can be solved in polynomial time via a simple greedy heuristic [33] and even $R||\sum C_j$ is in P via matching techniques [26].

Scheduling with incompatibilities has first been considered in the 1990's by Jansen, Bodlaender and Woeginger [22], who studied $P||C_{\max}$ with incompatibilities between jobs. Among other things, they present an approximation algorithm whose approximation ratio depends on the quality of a coloring for the incompatibility graph. The result yields constant-factor approximation algorithms for the subproblem where the incompatibility graph can be colored in polynomial time with a constant number of colors. Furthermore, Jansen and Bodlaender present hardness results in the setting of co-graphs, bipartite graphs and interval graphs [21].

More recently, there has been a series of results for uniformly related machines and unit processing times [63, 64, 127] for several classes of incompatibility graphs like (complete) bipartite graphs, forests, or k-chromatic cubic graphs. In 2012, Dokka, Kouvela, and Spieksma [43] presented approximation and inapproximability results for the so called multi-level bottleneck assignment problem. This problem can be seen as a variant of $P|\text{cliques}|C_{\max}$ in which each clique has the same size and each machine has to receive exactly one job from each clique. However, the exact setting studied in our paper was introduced only recently by Das and Wiese [42], who called the cliques *bags*. They obtained a PTAS for $P|\text{cliques}|C_{\max}$ and showed that (unless P equals NP) there is no constant-factor approximation algorithm for the restricted assignment variant $P|\text{cliques}, M(j)|C_{\max}$, i. e., the case in which each job j may only be processed on a given set $M(j)$ of machines eligible for j. Moreover, they gave an 8-approximation for the special case $P|\text{cliques}, M(k)|C_{\max}$ in which jobs belonging to the same clique have the same restrictions, i. e., sets $M(k)$ of eligible machines are given for each clique $k \in [c]$.

This line of research was continued by two groups. In particular, Grage, Jansen and Klein [74] obtained an EPTAS for $P|\text{cliques}|C_{\max}$, and Page and Solis-Oba [142] considered a variant of $R|\text{cliques}|C_{\max}$ where the number of machine kinds and cliques is restricted and obtained a PTAS among many

other results. Two machines are of the same kind if the processing time of each job is the same on both of them.

Regarding FPT algorithms for scheduling problems, a good overview on this line of research is provided in a survey by Mnich and van Bevern [133]. The most notable result in our context is probably a work due to Knop and Koutecký [115] who used n-fold integer programs to prove (among other things) two FPT results for $R||\sum w_j C_j$. In particular, $R||\sum w_j C_j$ is FPT with respect to the number of machines and the number of different job kinds ϑ. Further, it is also FPT with respect to the maximum processing time, the number of different job kinds ϑ, and the number of distinct machine kinds. These results were generalized and greatly extended by Knop et al. in [116]. In their work, they introduce a general framework for solving various configuration IPs by modeling them as (an extended version of) the Monoid Decomposition problem. This allows the solving of many problems with different kinds of objects (for example, jobs with release times and due dates) and locations (for example, unrelated machines) and (linear or non-linear) objectives in FPT with plenty different, natural parameterizations.

The results presented in this chapter are a subset of the paper [101]. In that work, we present various results regarding incompatibility cliques which results in a richer, more interesting picture. First, we give polynomial time algorithms to solve various variants optimally using matching techniques and dynamic programming. For example, we show that the problem $P|\text{cliques}|\sum C_j$ and the problem $R|\text{cliques}|\sum C_j$, in which jobs belonging to the same clique have the same processing times, are in P. This remains true even if we introduce additional job dependent assignment restrictions. Note that this setting is closely related to the case with clique dependent assignment restrictions introduced by Das and Wiese [42]. We study this case as well and prove it to be NP-compete and even APX-hard already for the case with only two different processing times. Further, scheduling with assignment restrictions for the cliques rather than the jobs, i. e., $R|\text{cliques}|\sum C_j$, is NP-hard and scheduling on unrelated machines becomes APX-hard.

STRUCTURE OF THIS CHAPTER In Section 7.1, we start with an investigation of the Scheduling problem under the objective of minimizing the total completion time with clique machine restrictions. We show that this problem is FPT parameterized by the number of cliques. Then, we turn our attention to the objective of minimizing the weighted total completing time. Section 7.2 proves that this problem is FPT parameterized by the number of machines, the largest processing time and the number of job kinds. In turn, we can omit that the number of machines has to be a parameter by exchanging it with the parameter number of cliques obtaining an alternative FPT result, see Section 7.3.

7.1 FEW CLIQUES AND CLIQUE MACHINE RESTRICTIONS

This section considers the problem variant $P|\text{cliques}, M(k)|\sum C_j$. Recall that in this setting, we have a set $M(k)$ of machines for each clique $k \in [c]$. In a feasible schedule, jobs of clique k are scheduled exclusively on machines $i \in M(k)$. We prove the following result:

Theorem 37. *The problem $P|\text{cliques}, M(k)|\sum C_j$ can be solved in FPT parameterized by the number of cliques c.*

To prove this result, we first establish some notation and basic observations. Then, we introduce an integer programming model with n-fold form, and lastly, argue that it can be solved efficiently.

In any schedule for an instance of this problem, there can be at most c jobs scheduled on each machine due to the clique constraints. Hence, we may imagine that there are *c slots* on each machine numbered in chronological order (the slot which is scheduled first has the lowest number). We further use the intuition that the slots form c layers with all the first slots in the first layer, all the second slots in the second one, and so on. Obviously, we can represent any schedule by an assignment of the jobs to these slots. Some of the slots may be empty, and we introduce the convention that all the empty slots (hence taking 0 time) on a machine should be in the beginning, i. e., intuitively speaking, dummy jobs with processing time 0 are scheduled at first. If a job of clique k is scheduled in a certain slot, we say that k is present in the slot, in the corresponding layer and on the machine. Having this notations and observations at hand, we can prove the theorem.

Proof. To obtain the desired schedule, we only need the pattern of cliques present on the machine and call such a pattern a *configuration*. More precisely, we call a vector $\kappa = (\kappa_1, \dots, \kappa_c) \in \{0, 1, \dots, c\}^c$ a configuration if the following two conditions are satisfied. Note that 0 represents an empty slot:

- $\forall \ell, \ell' \in [c] : \kappa_\ell = \kappa_{\ell'} \wedge \ell \neq \ell' \implies \kappa_\ell = \kappa_{\ell'} = 0$,
- $\forall \ell \in [c-1] : \kappa_\ell > 0 \implies \kappa_{\ell+1} > 0$.

The first condition corresponds to the requirement that at most one job of each clique should be scheduled on a machine. The second one matches to the convention that the empty slots are at the beginning.

We denote the set of configurations as $\mathcal{K}$. Moreover, for each $k \in [c]$, $\mathcal{K}(k)$ denotes the set of configurations in which k is present, i. e., $\mathcal{K}(k) = \{\kappa \in \mathcal{K} \mid \exists \ell \in [c] : \kappa_\ell = k\}$. Note that $|\mathcal{K}| \leq (c+1)!$ since there can be up to c zeros in a configuration and a configuration excluding the zeros can be seen as a truncated permutation of the numbers in $[c]$. We call a configuration κ *eligible* for a machine i if all the cliques occurring in κ are eligible on i, that is, for each $\kappa_\ell \neq 0$, we have $i \in M(\kappa_\ell)$.

A schedule for an instance of the problem trivially induces an assignment $\tau : \mathcal{M} \to \mathcal{K}$ of the machines to the configurations. In particular, identify the cliques of the jobs in the present order and fill 0 slots at the beginning

until c slots are occupied on that machine. Call such an assignment *feasible* if there exists a feasible schedule corresponding to τ. That is, if $\tau(i) = \kappa$ is eligible on i for each machine i and, for each clique k, the number of machines assigned to a configuration in $\mathcal{K}(k)$ is equal to the number of jobs in k. Note that different schedules may have the same assignment, i. e., when jobs of the same clique change positions.

Given such a feasible assignment τ, we can find a schedule corresponding to τ with a minimal objective function value via a simple greedy procedure. Namely, for each clique k, we can successively choose a smallest job that is not scheduled yet and assign it to a slot positioned in the lowest layer that still includes non-empty slots belonging to k according to τ. This is indeed optimal, as a job in slot ℓ contributes $c - \ell + 1$ times to the total completion time. Exchanging this job j with processing time $p_j < p_{j'}$ with another job j' at some position $\ell' > \ell$ from the same clique would yield an increase in the sum of completion times. We reduce the value by $(\ell' - \ell)p_j$ by shifting j up, but in turn, gain $(\ell' - \ell)p_{j'}$. As $p_j < p_{j'}$, this worsens the value of the objective function.

As explained above, we can associate an objective value to each feasible assignment. In the next step, we introduce an n-fold integer program to search for such a feasible assignment τ with minimal objective.

We introduce two types of variables, that is, $x_{\kappa,i} \in \{0,1\}$ for each machine $i \in \mathcal{M}$ and configuration $\kappa \in \mathcal{K}$ corresponding to the choice of whether i is assigned to κ or not. Further, we ensure $x_{\kappa,i} = 0$ if κ is not eligible on i using the restrictive upper bound 0 in that case. Further, we have $y_{k,\ell,i} \in \{0,1,\dots,N\}$ for each clique $k \in [c]$ and layer $\ell \in [c]$ counting the number of slots reserved for clique k in the layers 1 to ℓ. The duplication for each machine $i \in \mathcal{M}$ is only a technical detail to obtain the n-fold structure and does not carry any further meaning. To use this variable correctly, we choose some arbitrary machine $i^* \in \mathcal{M}$ and set $y_{k,\ell,i} = 0$ for each $i \neq i^*$ using lower and upper bounds for the variables. Let $\mathcal{K}(k,\ell) = \{\kappa \in \mathcal{K} \mid \exists \ell' \in [\ell] : \kappa_{\ell'} = k\}$ for each $k, \ell \in [c]$ be the set of configurations where clique k is present in one of the first ℓ layers. Let n_k be the number of jobs belonging to clique k, and $p_{k,q}$ the size of the job that has position q if we order the jobs of clique k non-decreasingly by size. The integer program has the following form:

$$\min \sum_{\ell=1}^{c} \sum_{k=1}^{c} \sum_{q=1}^{y_{k,\ell,i^*}} p_{k,q}$$

$$\sum_{i \in \mathcal{M}} \sum_{\kappa \in \mathcal{K}(k,\ell)} x_{\kappa,i} = \sum_{i \in \mathcal{M}} y_{k,\ell,i} \qquad \forall k \in [c], \ell \in [c] \tag{1}$$

$$\sum_{i \in \mathcal{M}} y_{k,c,i} = n_k \qquad \forall k \in [c] \tag{2}$$

$$\sum_{\kappa \in \mathcal{K}} x_{\kappa,i} = 1 \qquad \forall i \in \mathcal{M} \tag{3}$$

Constraint (3) ensures that exactly one configuration is chosen for each machine; due to (1), the variables $y_{k,\ell,i}$ correctly count the slots reserved

for clique k; and (2) guarantees that the jobs of each clique are covered. The IP is indeed of n-fold structure where Constraint (3) is locally uniform and duplicated for all machines and the remaining constraints are globally uniform.

Observe that if there is an (optimal) schedule, the n-fold IP has a solution as argued above, i. e., we can find the assignment τ and by that count the values for the variables. By the greedy approach described before, we can also derive a schedule from that solution. First, assign the configurations implied by the $x_{\kappa,i}$ variables, reserve the slots for each clique by the $y_{k,\ell,i}$ variables and finally, fill them greedily by the corresponding (smallest) jobs. Thus, if there is no solution to the above IP, the instance is not feasible.

Concerning the objective function, for each clique k, we sum up the smallest $y_{k,1,i^*}$ job sizes for the first layer, the smallest $y_{k,2,i^*}$ sizes in the second one, and so on. Note that this counting is correct since we use the convention that empty slots are at the beginning and therefore, each job contributes once to the objective for its own layer and once for each following layer.

This objective function is indeed separable convex: Note that many of the variables do not occur in the objective and hence, can be ignored in the following. We essentially have to consider the function $g_k : [n_k] \to \mathbb{R}$ with $g_k(y_{k,\ell,i^*}) = \sum_{q=1}^{y_{k,\ell,i^*}} p_{k,q}$ for each $k \in [c]$ since the objective can be written as $\sum_{\ell,k\in[c]} g_k(y_{k,\ell,i^*})$. Let $\{x\} = x - \lfloor x \rfloor$ for each $x \in \mathbb{R}$ and $\tilde{g}_k : \mathbb{R} \to \mathbb{R}$ with:

$$\tilde{g}_k(x) = \begin{cases} p_{k,1}x & \text{if } x < 1 \\ p_{k,\lceil x\rceil}\{x\} + \sum_{q=1}^{\lfloor x\rfloor} p_{k,q} & \text{if } \lfloor x\rfloor \in [n_k - 1] \\ p_{k,n_k}(x - n_k) + \sum_{q=1}^{n_k} p_{k,s} & \text{if } x \geq n_k \end{cases}.$$

Then, we have $\tilde{g}_k(y_{k,\ell,i^*}) = g_k(y_{k,\ell,i^*})$ for each $k \in [c]$, as for each integral value $\{x\}$, or respectively $x - n_k$ for $x = n_k$ is zero, as well as $\lfloor x \rfloor = x$ and thus, solely the desired sum remains. Furthermore, $\tilde{g}_k$ is continuous and essentially a linear function with $n_k - 1$ points at which the slope changes. Due to the ordering of the processing times, the slope can only increase and hence, the function is convex.

Finally, in order to use Proposition 36, we have to estimate the parameters:

- $n = M$,
- $t = O((c+1)!) = O(c^c)$,
- $r = c^2 + c = O(c^2)$,
- $s = 1$,
- $\Delta = 1$,
- $\log(\|u - \ell\|_\infty) = O(\log(N))$,
- $\log(f_{max}) = O(\log(p_{\max} c^2 N)) = O(\log(N c p_{\max}))$.

Applying Proposition 36 for solving the ILP gives a running time of $2^{O(c^4 \log(c))} M \log(M) \log(N) \log(p_{\max})$. Obtaining the schedule as described above, we go through each of the cM slots and assign the smallest job greedily. This takes time $O(cMN \log(N))$, yielding an overall running time of $2^{O(c^4 \cdot \log(c))} MN \log(M) \log(N) \log(p_{\max})$, finishing the proof. □

7.2 FEW MACHINES AND WEIGHTS

In this section, we consider the problem of scheduling jobs non-preemptively on unrelated machines with clique incompatibility under the objective to minimize the sum of weighted completion times, i. e., $R|\text{cliques}|\sum w_j C_j$.

But let us first introducing *kinds* of jobs formally. Two jobs j and j' belong to the same kind if their processing time vectors $(p_j^1, \ldots, p_j^M)$ and $(p_{j'}^1, \ldots, p_{j'}^M)$ and their weights w_j and $w_{j'}$ are equal. Denote the number of *job kinds* as ϑ. We can re-write the set of jobs as $(n_{1,1}, \ldots, n_{\vartheta,1}, n_{1,2}, \ldots, n_{\vartheta,k})$ where jobs of kind j and from clique k appear $n_{j,k}$ times. Denote by $p_{\max}$ the largest processing time and by $w_{\max}$ the largest weight occurring in the instance. In the remaining of this section, we prove the following theorem:

Theorem 38. *The problem $R|\text{cliques}|\sum w_j C_j$ can be solved in FPT parameterized by the number of machines M, the largest processing time $p_{\max}$ and the number of job kinds ϑ.*

Similar to before, we aim to model the problem as an n-fold IP. The main obstacle to do so is to formulate an appropriate objective function. In [115], Knop and Koutecký developed a quadratic separable convex function equivalent to the sum of completion times objective. This result relies on the fact that in an optimal schedule, the jobs on each machine are ordered regarding the Smith's rule, i. e., the jobs are schedules non-increasingly regarding $\phi_i(j) = w_j / p_j^i$ [71]. We may visualize this as a Gantt chart for each machine: Roughly speaking, it is a line of neighboring rectangles in the order of the schedule. The width of the jth rectangle is the processing time of the jth job on the machine and the rectangles height corresponds to the total weight of all uncompleted jobs (thus including the jth job). The area under the function, i. e., an integral of the weights of uncompleted jobs in time, corresponds to the weighted completion time and can be separated into two parts. One part is dependent only on the job kind and machine. The second one is dependent on the composition of the jobs assigned to the machine. By the fact that for any machine the Smith's order is optimal, the order of job kinds is known. Hence, the composition is determined by the number of jobs of each kind assigned to the machine. Thus, the second part yields a piece-wise linear convex function. For details, see [115]. Altogether, they prove:

Proposition 39 ([115]). *Let $x_1^i, \ldots, x_\vartheta^i$ be numbers of jobs of each kind scheduled on a machine i and let $\pi_i \colon [\{1, \ldots, \vartheta\}] \to [\{1, \ldots, \vartheta\}]$ be a permutation of job kinds such that $\phi_i(\pi_i(j)) \geq \phi_i(\pi_i(j+1))$ for all $1 \leq j \leq \vartheta - 1$. Then, the contribution of i to the weighted completion time in an optimal schedule is*

equal to $\sum_{j=1}^{\vartheta}(1/2(z_j^i)^2(\phi_i(\pi_i(j))-\phi_i(\pi_i(j+1)))+1/2\cdot x_j^i p_j^i w_j)$ *where* $z_j^i=\sum_{\ell=1}^{j} p^i_{\pi_i(\ell)}x^i_{\pi_i(\ell)}$.

Using this, we can now prove Theorem 38.

Proof. First, let us focus on constructing the n-fold IP. For this result, we extend the n-fold IP introduced in [115] and adapt the separable convex function to our needs. Note that in [116], these results are generalized, but by that also more complex. Further, using these results does not improve upon our running times.

Even though the authors separate their constraints into globally uniform and locally uniform ones in [115], the overall number of constraints is only dependent on the parameters. Thus, we can shift their second constraint to the globally uniform A_i blocks and incorporate the clique constraints as locally uniform ones. There, we ensure that all jobs are covered and that each machine schedules at most one job from each clique where each locally uniform B_i block covers one clique. Denote by $\pi_i(j)$ for $j\in[\vartheta]$ the permutation of jobs of the jth kind according to any fixed Smith's ordering of the kinds on machine i. Let $x^i_{j,k}$ be a variable that corresponds to the number of jobs of kind $j\in[\vartheta]$ from clique $k\in[c]$ that are scheduled on machine $i\in\mathcal{M}$. Consider the following IP

$$\sum_{k=1}^{c}\sum_{\ell=1}^{j} x^i_{\pi_i(\ell),k}p^i_{\pi_i(\ell)} = z_j^i \qquad \forall j\in[\vartheta],\forall i\in\mathcal{M} \tag{1}$$

$$\sum_{i=1}^{M} x^i_{j,k} = n_{j,k} \qquad \forall j\in[\vartheta],\forall k\in[c] \tag{2}$$

$$\sum_{j=1}^{\vartheta} x^i_{j,k} \leq 1 \qquad \forall i\in\mathcal{M},\forall k\in[c] \tag{3}$$

with lower bounds 0 for all variables and upper bounds $x^i_{j,k}\leq 1$ or respectively $x^i_{j,k}\leq 0$ if jobs of kind j cannot be scheduled on machine i, and $z_j^i\leq c\cdot p_{\max}$.

Let the $x^i_{j,k}$ variables form a vector $\mathbf{x}$ and the z_j^i variables from a vector $\mathbf{z}$. Denote by $\mathbf{x}^i$ and $\mathbf{z}^i$ the corresponding subset restricted to one machine i. The objective is to minimize the function $f(\mathbf{x},\mathbf{z})=\sum_{i=1}^{M} f^i(\mathbf{x}^i,\mathbf{z}^i)=\sum_{i=1}^{M}\sum_{j=1}^{\vartheta}(1/2(z_j^i)^2(\phi_i(\pi_i(j))-\phi_i(\pi_i(j+1)))+1/2\sum_{k=1}^{c} x^i_{j,k}p_j^i w_j)$. As we consider the altered variables $x^i_{j,k}$ over all cliques simultaneously, this corresponds to the objective function from Proposition 39. Thus, the function expresses the sum of completion times objective. Further, it obviously stays separable convex.

Regarding the constraint matrix, Constraint (1) is satisfied if the z_j^i variables are set as demanded in Proposition 39, i. e., the jobs are scheduled with respect to the Smith's rule. Constraint (2) ensures that the number of jobs from a kind j and clique k scheduled on the machines matches the overall number

of jobs from that kind and clique. Finally, Constraint (3) ensures that the number of jobs scheduled on a machine i from the same clique k is at most one.

We construct a schedule from the solution of the above IP in the following way: Place the jobs accordingly to the $x^i_{j,k}$ variables and the Smith's rule. That is, assign $x^i_{j,k}$ jobs of job kind j from clique k to machine i (note that this number is at most one due to Constraint (3)). After assigning all jobs to a machine, place them non-increasingly regarding the Smith's ratio $\phi_i(j)$ onto the machine. This takes time $O(N \log(N))$, as we go through all jobs once and sort them afterwards. As we did not change the objective from [115], such a solution corresponds to an optimal one regarding the sum of weighted completion times objective.

Regarding the running time for the n-fold IP, we first have to estimate its parameters. The first constraint is globally uniform whereas the second and third constraints are locally uniform and repeated for each clique. The parameters can be bounded by

- $n = c$,
- $t = O(\vartheta \cdot M)$,
- $r = O(\vartheta \cdot M)$,
- $s = O(\vartheta + M)$,
- $\Delta = p_{\max}$,
- $\log(\|u - \ell\|_\infty) = \log(c \cdot p_{max})$,
- $\log(f_{max}) = O(\log(M \cdot c^2 \cdot p_{\max} \cdot w_{\max})) \leq O(\log(Mcp_{\max} w_{\max}))$.

Applying Proposition 36 yields a running time of $(p_{\max}\vartheta M)^{O(\vartheta^3 M^3)} O(c \log^3(c) \log(w_{\max}))$. Note that the inequality constraints do no harm, as we can introduce parameter many slack-variables to turn them into equality constraints. Asymptotically, this does not influence the running time. Then, we build the schedule in time $O(N \log(N))$ yielding an overall running time of $(p_{\max}\vartheta M)^{O(\vartheta^3 M^3)} O(c \log^3(c) \log(w_{\max})) + N \log(N)$. □

7.3 FEW CLIQUES AND WEIGHTS

Let us turn our attention to the same problem $R|\text{cliques}|\sum w_j C_j$ but parameterized by c, $p_{\max}$ and ϑ. Let the definitions be as in the previous section. The following n-fold IP is an extended formulation of the one from [115]. However, the authors did not consider cliques, thus we embed them appropriately. This leads to the following theorem:

Theorem 40. *The problem $R|\text{cliques}|\sum w_j C_j$ can be solved in FPT parameterized by the number of cliques c, the largest processing time $p_{\max}$ and the number of job kinds ϑ.*

Proof. Regarding the variables for our IP, let again $x^i_{j,k}$ denote that $x^i_{j,k}$ jobs of kind $j \in [\vartheta]$ from clique $k \in [c]$ are scheduled on machine $i \in \mathcal{M}$. Further, as before, we have z^i_j for each $j \in [\vartheta]$ and $i \in \mathcal{M}$. Denote by $\pi_i(j)$ the permutation of job kinds with respect to the machines. Consider the following IP

$$\sum_{i=1}^{M} x^i_{j,k} = n_{j,k} \qquad \forall j \in [\vartheta], \forall k \in [c] \tag{1}$$

$$\sum_{k=1}^{c} \sum_{\ell=1}^{j} x^i_{\pi_i(\ell),k} p^i_{\pi_i(\ell)} = z^i_j \qquad \forall j \in [\vartheta], \forall i \in \mathcal{M} \tag{2}$$

$$\sum_{j=1}^{\vartheta} x^i_{j,k} \leq 1 \qquad \forall k \in [c], \forall i \in \mathcal{M} \tag{3}$$

with lower bounds 0 for all variables and upper bounds $x^i_{j,k} \leq 1$ or respectively $x^i_{j,k} \leq 0$ if jobs of kind j cannot be scheduled on machine i, and $z^i_j \leq c \cdot p_{\max}$. The objective function is to minimize $f(\mathbf{x},\mathbf{z}) = \sum_{i=1}^{M} f^i(\mathbf{x}^i,\mathbf{z}^i) = \sum_{i=1}^{M} \sum_{j=1}^{\vartheta} (1/2(z^i_j)^2(\phi_i(\pi_i(j)) - \phi_i(\pi_i(j+1))) + 1/2 \sum_{k=1}^{c} x^i_{j,k} p^i_j w_j)$. As before, we altered the x^i_j variable in the objective function by introducing more indices. However, as we only consider the sum of these variables, this does not affect the objective. Thus, by Proposition 39, the function maps correctly to the sum of weighted completion times objective.

Regarding the IP, the constraints resemble the ones from previous IP. Constraint (1) is satisfied if the number of jobs from kind j and clique k are covered by the number of jobs from that kind and clique scheduled on the machines. Further, Constraint (2) is satisfied if the variable z^i_j is set accordingly to Proposition 39, i. e., the jobs are scheduled with respect to the Smith's rule. The last constraint is the same as in the previous IP and ensures that the number of jobs scheduled on machine i from the same clique k is at most one.

A solution to the n-fold IP can be transformed into a schedule by placing $x^i_{j,k}$ jobs of job kind j and from clique k onto machine i (again, this is at most one job due to Constraint (3)) and ordering the jobs non-increasingly regarding the Smith's ratio $\phi_i(j)$, again taking $O(N \log(N))$ time.

To finally argue the running time of the IP, let us estimate the parameters. The first constraint is globally uniform. The remaining ones are locally uniform and repeated for each machine. We can bound the parameters by:

- $n = M$,
- $t = O(\vartheta \cdot c)$,
- $r = O(\vartheta \cdot c)$,
- $s = O(\vartheta + c)$,
- $\Delta = p_{\max}$,

- $\log(\|u-\ell\|_\infty) = \log(c \cdot p_{max})$,
- $\log(f_{max}) = O(\log(M \cdot (c \cdot p_{\max}^2 + c \cdot p_{\max} \cdot w_{\max})))$.

Applying Proposition 36 and building the schedule yields a running time of $(c\vartheta p_{\max})^{O(\vartheta^3 c^3)} O(M \log^2(M) \log(w_{\max})) + N \log(N)$. Again the inequality constraints in the IP do no harm, as we can introduce few slack-variables to turn them into equality constraints. Asymptotically, this does not influence the running time. □

Let's drop rocks on them!

— from *Eragon* by Christopher Paolini

8

CLASS CONSTRAINT SCHEDULING: CONSTANT-FACTOR APPROXIMATION

This chapter studies another natural extension of the classical Scheduling problem with respect to the makespan minimization objective, called Class Constrained Scheduling (CCS). In this problem, each job additionally admits a class and each machine can only schedule jobs from a limited number of different classes. This restriction arises commonly when considering problems such as product planning or data placement: Often times, operations need access to databases or specific operating tools. Due to logistical limitations or time restrictions, these databases and tools have to be stored locally. However, space and storage is limited. Hence, not all requirements for all jobs can be stored on a machine. Thus, we have to guarantee that for each job scheduled on a machine, we are able to store all its requirements. Further application examples like, e.g., distributing students in a university, crew scheduling in airline industries, or the above mentioned production planning are discussed in [30, 153]. Moreover, note that class constraints are a natural generalization of cardinality constraints (each job admits a different class), which are studied for a wide range of combinatorial optimization problems and have many applications [25].

Formally, we are given a set $\mathcal{J}$ of N jobs, where each job $j \in \mathcal{J}$ has a processing time $p_j \in \mathbb{N}$ and some class $c_j \in \{1,\dots,C\}$. These jobs have to be allotted onto a set $\mathcal{M}$ of M identical machines, each with a limitation of c class slots. A *class slot* can contain any number of jobs from one arbitrary class. Clearly, we can assume that $C \leq N$, as classes without any belonging job can be discarded. Further, $c \leq C$, as otherwise, we allow more classes to be scheduled on one machine than classes exist overall. Hence, in both cases, all jobs can be placed feasibly on a single machine yielding the classical Scheduling problem.

The input can be described as an instance $I = [p_1,\dots,p_N, c_1,\dots,c_N, M, c]$. The encoding length of such an instance is given by

$$\begin{aligned}|I| &= O\Big(\sum_{j=1}^{N} \lceil \log(p_j) \rceil + \sum_{j=1}^{N} \lceil \log(c_j) \rceil + \lceil \log(M) \rceil + \lceil \log(c) \rceil\Big) \\ &\leq O(N \log(p_{\max}) + N \log(N) + \log(M)).\end{aligned}$$

Note that c and C are dominated by N and thus, do not appear in the big O notation.

In the following, we study each case of feasible job placement, i. e., the non-preemptive, the splittable and the preemptive case. Recall that in the non-preemptive case, we have to place all jobs as a whole. In the splittable

case, we are allowed to cut the jobs into arbitrarily small pieces and place them anywhere as long as we do not schedule two jobs at the same time on the same machine. Lastly, in the preemptive case, we can also split the jobs but pieces of the same job are not allowed to be scheduled in parallel, i. e., at the same time on different machines. The output for each case is described by a schedule σ which describes a mapping from jobs or job pieces to the corresponding machines and, if applicable, starting times. For formal definitions of the respective schedule, we refer to the Chapter 2.

Note that in the splittable case as well as in the preemptive case, the output length may be huge compared to the input, as we can produce arbitrarily many job pieces. Furthermore, the number of machines (which only appears logarithmically in the input) and thus, the number of explicit job allotments may be a lot larger than the number of jobs. However, we managed for all algorithms to bound the output length by a polynomial in the encoding length either by carefully encoding the output or by bounding the number of produced job pieces appropriately.

In the following, we aim to find a schedule σ such that the makespan is minimized while assigning jobs of at most c different classes onto a machine. We denote an optimal schedule with minimum makespan by $\textsc{opt}(I)$. Finding $\textsc{opt}(I)$ is well-known to be NP-hard for all cases [54, 153]. Thus, we are satisfied to produce a solution close to the makespan of an optimal schedule.

In this chapter, we present simple constant-factor approximation algorithms which produce a solution σ efficiently. For the non-preemptive case, we guarantee a quality of $\mu(\sigma) \leq 7/3 \cdot \mu(\textsc{opt}(I))$. The algorithm runs in time $O(N^2 \log(N) + N \log^2(p_{\max}))$. Adjusting the algorithm to deal with the splitting and the non-parallelism yields a ratio of 2 for the splittable and the preemptive case. The algorithms for the splittable case with $M \leq N$ and for the preemptive case run in time $O(N^2 \log(N))$. If $M > N$ holds for the splittable case, we can encode the instance of each step and the output efficiently and by that, obtain a running time of $O(N^2 \log(M))$.

Each of the algorithms shares the same, simple framework with only minor adaptions for the distinct cases: (1) Guess the optimal makespan T; (2) group the jobs belonging to classes with an overall processing time larger than T into as few new classes as possible, each smaller than T; (3) distribute all classes via round robin, a cyclic scheduling approach where the number of used class slots on each machine is balanced. Overall, this approach, embedded into a binary search for the optimal makespan, guarantees to find a feasible schedule, if one exists. To the best of our knowledge, these are the first algorithms to produce solutions with a constant approximation ratio for these problems.

SUMMARY OF RESULTS

- We establish a $7/3$-approximation algorithm for the non-preemptive case with running time $O(N^2 \log(N) + N \log^2(p_{\max}))$.
- Further, we develop a 2-approximation algorithm for the splittable case with running time $O(N^2 \log(N))$ if $M \leq N$.

- We adjust the algorithm for the splittable case to also handle a large number of machines (i.e., $M > N$) by encoding the instance (of each step) and the output efficiently. This yields a 2-approximation algorithm with running time $O(N^2 \log(M))$ if $M > N$.
- Adapting the algorithm from the splittable case to handle the parallelism of job pieces, we get a 2-approximation algorithm for the preemptive case with running time $O(N^2 \log(N))$.

FURTHER RELATED WORK For CCS, approximation schemes are known for two special cases. On the one hand, Shachnai and Tamir [153] presented a PTAS for the case that the number of classes C is a constant. On the other hand, Chen et al. [30] designed a PTAS for the case that each class contains exactly one job. The latter result even works if the class constraints are machine dependent, that is, for each machine i a class constraint k_i is given.

After this work was published first, inspired by the above mentioned generalization, we investigated if our constant-factor approximation algorithms can be adapted to also handle machine dependent class constraints. We answered this question affirmatively, yielding nearly the same approximation ratios and running times [160].

For the remainder of this section, we discuss the known results for the class constrained versions of Bin Packing (CCBP) and Multiple Knapsack (CCKP). These problems are closely related to CCS and NP-hard as well [54, 152]. Thus, studying approximation schemes is a natural approach to obtain satisfactory solutions efficiently.

Golubchik et al. present in [72] a PTAS for CCKP for the case that all knapsacks are identical. If C is a constant, there is also a PTAS for the general case by Shachnai and Tamir introduced in [153]. Furthermore, they present an FPTAS for 0-1 CCKP when all items are distinct in the same work [153], i. e., a PTAS where the running time is also polynomial in $1/\epsilon$. However, the problem becomes APX-hard when each item admits a set of classes [153]. Thus, there is no PTAS unless P = NP.

Xavier et al. prove in [166] that CCBP admits an APTAS if C is a constant, i. e., a PTAS with an additional additive error. Furthermore, there is no APTAS when c is not constant [54]. Improving upon the APTAS of Xavier et al., Epstein et al. [54] present an AFPTAS for the case of a constant C. An AFPTAS is an APTAS where the running time is not only polynomial in the number of items but also in $1/\epsilon$. A special sub-case for CCBP is the cardinality constrained version where $C = N$ and thus, the bound on the number of classes on a bin corresponds to the bound on the number of items. This problem admits an APTAS [28] and an AFPTAS [57].

STRUCTURE OF THIS CHAPTER We structure the remainder of this chapter by the respective cases. In Section 8.1, we develop an approximation algorithm for the non-preemptive CCS problem. Then, in Section 8.2, we adapt this algorithm to handle the splittable case. This section also presents the approach to handle the case of unbounded many machines to obtain the

desired running time. Finally, Section 8.3 presents the adjustments for the preemptive case to handle the parallelism of job pieces.

8.1 NON-PREEMPTIVE CASE

We start with the non-preemptive case where we have to place each job as a whole. Let the overall running time P_u of a class $u \in [C]$ be the accumulated processing time of all its jobs, i. e., $P_u = \sum_{\{j \mid c_j = u\}} p_j$. Further, denote by C_u some lower bound on the number of machines that have to receive a job of class u in a schedule of some given makespan. How exactly this value is determined is described below. The framework for solving the non-preemptive case is displayed in Algorithm 1.

Input: Processing times $p_1, \ldots, p_N$,
corresponding classes $c_1, \ldots, c_N$,
number of machines M, class slot restriction c

Calculate P_u for each class u.
Calculate the lower bound $LB = \max\{\sum_{j=1}^{N} p_j / M, p_{\max}\}$ and the
  upper bound $UB = c \cdot \max_u\{P_u\}$.
Do a binary search between LB and UB, where T is the current guess:
  Restore the set of original classes.
  For each class with $P_u > T$: Compute C_u and divide the class into
    C_u new, unique classes, each new class u' with $P_{u'} \leq T$.
  Delete the original classes $u \in [C]$ with $P_u > T$.
  Denote by C' the number of current classes, i. e., the number of classes
    generated by grouping and the original classes with $P_u \leq T$.
  If $C' > c \cdot M$: Increase the current guess T.
  Otherwise:
    Save the present (grouped and small) classes.
    Lower the guess T.
Compute $P_{u'}$ for $u' \in [C']$.
Sort the present classes in non-ascending order regarding $P_{u'}$.
Allot the classes in non-ascending order regarding $P_{u'}$ onto the
  machines via round robin. (*)
Reassign the original classes to the jobs.
Output schedule.

Algorithm 1: Framework for solving the CCS problem. The label (*) serves as an anchor for the changes when handling the preemptive case.

The algorithm searches for the optimal makespan via a binary search. First of all, in each iteration, the original classes are restored (only necessary after the first iteration). Then, the classes with $P_u > T$ are divided into C_u new,

unique classes, each with an accumulated processing time smaller or equal to $(4/3)T$. To do so, we first have to compute an appropriate lower bound on the number of machines class u needs in an optimal schedule, i. e., the number of class slots occupied by this class. This lower bound consists of two estimations: First, we compute the needed class slots regarding the area, i. e., the least number of machines $C_u^{(1)} = \lceil P_u / T \rceil$ necessary to schedule the overall processing time of the class with respect to the makespan guess T. Secondly, we also compute a lower bound $C_u^{(2)}$ of needed class slots for jobs with $p_j > (1/3)T$. Then we take the maximum of both values, i. e., $C_u = \max\{C_u^{(1)}, C_u^{(2)}\}$.

To compute $C_u^{(2)}$, we first count the number of jobs with $p_j > (1/2)T$. In the following, we call these jobs *huge* and the set of huge jobs $\mathcal{J}_{\text{huge}}$. Huge jobs have to be placed onto different machines to not exceed T in an optimal schedule. On top of them, we would like to place as many jobs with $(1/2)T \geq p_j > (1/3)T$ as possible. Denote these jobs as *medium* and the set of medium jobs as $\mathcal{J}_{\text{medium}}$. Placing the medium jobs can be done greedily by assigning the largest fitting job $j \in \mathcal{J}_{\text{medium}}$ which is not assigned yet on top of a huge one (considered in an arbitrary order). Denote with $\mathcal{J}'_{\text{medium}}$ the set of still unassigned medium jobs after this placement. Dividing $|\mathcal{J}'_{\text{medium}}|$ by 2, we get a lower bound on the number of machines needed to schedule the jobs from $\mathcal{J}'_{\text{medium}}$. Together, this yields $C_u^{(2)} = |\mathcal{J}_{\text{huge}}| + |\mathcal{J}'_{\text{medium}}|/2$.

Note that until now, we just calculated a lower bound on the necessary number of machines to schedule some class u with respect to the current guess T. However, we still have to actually group this class into C_u new classes. This is done by using the LPT algorithm. This algorithm assigns the jobs in non-increasing order regarding their processing time p_j such that a job j is placed onto the machine (or class in this context) with the lowest load at the current moment [75]. This yields that each new class u' has $P_{u'} \leq (4/3)T$ as proven in the next theorem.

Next, the classes which were split by grouping their jobs are deleted, i. e., all original classes with $P_u > T$. Denote by C' the number of current classes, i. e., the number of new classes originating from the grouping and the original classes with $P_u \leq T$. The algorithm checks whether C' exceeds $c \cdot M$. If so, we discard the guess, as we cannot schedule more classes on M machines each with c class slots. In the other case, the current set of classes is saved and the guess on the makespan is lowered. The binary search terminates when the lowest feasible guess T satisfying this condition is calculated.

After we found the lowest feasible guess on the makespan, the current classes are distributed among the machines via *round robin*. Round robin is a procedure where the classes are placed in non-ascending order regarding $P_{u'}$ with $u' \in [C']$, such that the first class is assigned onto the first machine, the second one onto the second machine and so on until all machines admit a class. This is then repeated until all classes are assigned. An example is given in Figure 8.1. Note that this procedure is independent of the guess T. However, we prove in Theorem 42 that this algorithm indeed computes a feasible 7/3-approximation for the non-preemptive case of the Class Constrainted

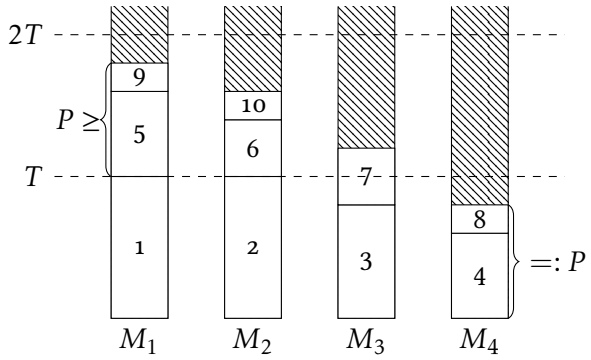

Figure 8.1: The figure shows an example of the round robin scheduling approach. The classes are sorted and numbered regarding their total processing times. Further, it is highlighted that the load L'_1 of machine M_1 when removing its largest job $j = 1$ satisfies $L'_1 \leq L_4 = L'_4 = P$.

SCHEDULING problem. But first, let us prove a general lemma regarding round robin, which comes in handy when proving the approximation ratio of our algorithms.

Lemma 41. *Let σ be a schedule produced by round robin when packing N jobs with processing times $p_1, \ldots, p_N$ onto M machines. Then $\mu(\sigma) \leq \sum_{j=1}^{N} p_j/M + p_{\max}$.*

Proof. We prove this lemma by contradiction: Consider the loads L_i for each machine $i \in \{1, 2, \ldots, M\}$. Assume there is one machine i^* with $L_{i^*} > \sum_{j=1}^{N} p_j/M + p_{\max}$. We show that under this assumption each machine has a load greater than $\sum_{j=1}^{N} p_j/M$ and hence, we get the contradiction that $\sum_{j=1}^{N} p_j = \sum_{i=1}^{M} L_i > M \sum_{j=1}^{N} p_j/M = \sum_{j=1}^{N} p_j$. If we remove the largest job j placed on i^*, then i^* still has a load of at least $\sum_{j=1}^{N} p_j/M$. W. l. o. g., we may assume that j was placed in the first iteration of round robin. Let L'_i be the load any machine i receives after j was placed. We have $L'_{i+1} \geq \cdots \geq L'_M \geq L'_1 \geq \ldots L'_i$ since the jobs are assigned in decreasing order with respect to their processing times. This is graphically displayed in Figure 8.1. Hence, we have $L_i \geq L'_i > \sum_{j=1}^{N} p_j/M$ for each machine i, completing the proof. □

Theorem 42. *The approximation algorithm above produces a solution σ with $\mu(\sigma) \leq (7/3) \cdot \mu(\text{OPT}(I))$ for the non-preemptive case in time $O(N^2 \log(N) + N \log^2(p_{\max}))$.*

Proof. Clearly, $LB = \max\{\sum_{j=1}^{N} p_j/M, p_{\max}\}$ and $UB = c \cdot \max_u\{P_u\}$ are the correct lower and upper bounds for the optimal makespan. The lower bound considers an equal distribution of the overall processing time as well as scheduling the largest job on one machine. The upper bound estimates the processing time of the c largest classes, which is the largest load one machine can get in a feasible schedule.

In each iteration of the binary search, we first restore the original set of classes and thus, consider the original input instance. The (original) classes u with $P_u > T$ are then grouped into C_u new ones, as they obviously do not fit onto a single machine. To estimate C_u, we consider the value $C_u^{(1)} = \lceil P_u / T \rceil$ corresponding to the area needed to distribute the overall load of class u with respect to the current guess T. Further, we bound the number $C_u^{(2)}$ of machines needed to schedule jobs with $p_j > (1/3)T$ and take the maximum of both values. To calculate $C_u^{(2)}$, we first distributing huge jobs with $p_j > (1/2)T$, as they cannot be scheduled together on one machine without exceeding T; then, add the largest fitting medium job with $p_j > (1/3)T$ greedily, which is optimal by a simple switching argument; and, as at most two of the remaining medium jobs $\mathcal{J}'_{\text{medium}}$ fit on a machine, we divide their number by 2. Thus, $C_u^{(2)} = |\mathcal{J}_{\text{huge}}| + |\mathcal{J}'_{\text{medium}}|/2$ yields a correct bound on the number of machines to schedule them.

Finally, using the LPT algorithm to group the jobs of class u into C_u new classes, we get a schedule with makespan at most $T + (1/3)T = (4/3)T$. By $C_u^{(2)}$, we get that we can place all jobs with $p_j > (1/3)T$ without exceeding the makespan of T on a machine. Hence a machine is only overpacked by one job j with $p_j \leq (1/3)T$. Note that it cannot be overpacked by more than one job, as this implies that all machines are already overpacked contradicting the area argument $C_u^{(1)}$.

Next, we delete all classes which were grouped, i. e., all original classes with $P_u > T$. This is correct, as we already handle their jobs by considering the new classes. The current guess of the binary search is successful if $C' \leq c \cdot M$, i. e., the number of class slots is sufficient for the present classes. For the lowest guess T satisfying this property, it holds that $T \leq \mu(\text{OPT}(I))$. This is true because we calculate the absolute minimum of machines and thus, class slots needed for each original class $u \in [C]$ to be scheduled with makespan T. Thus, if $C' > c \cdot M$, there is no schedule with makespan T.

After we found the lowest guess T on the makespan, we distribute the current set of classes via round robin. This approach allots each of the new classes and the original small ones as a whole. Hence, we do not exceed the class slot restriction. In the last step, we reassign the original classes via a simple mapping. Reinserting the original classes can only decrease the number of used class slots. This occurs when two new classes map to the same original class. Further, as we use exactly the same space as reserved for the new class, the load is not increased. Using Lemma 41, we get an overall makespan of $\sum_{j=1}^{N} p_j / M + \max\{P_{u'} \mid u' \in [C']\} \leq LB + (4/3)T \leq T + (4/3)T = (7/3)T \leq (7/3)\mu(\text{OPT}(I))$ yielding the desired approximation ratio.

Regarding the running time, we first compute all P_u and the lower and upper bound in time $O(N)$, as we have to add the processing time of each job exactly once. Calculating C_u and using the LPT algorithm can be done in time $O(N \log(N))$, as we have to sort and then distribute the jobs of each class. The binary search needs $O(\log(N \cdot p_{\max})$ iterations due to our estimation

of the upper bound. After finding the lowest feasible guess on T, we sort all classes in time $O(cM \log(cM))$ and distribute them in time $O(cM)$ by round robin. Lastly, we reassign the correct classes via a simple mapping to the corresponding jobs, again in time $O(cM)$. This results in an overall running time of $O(N) + O(\log(N \cdot p_{\max})) \cdot O(N \log(N)) + O(NM \log(NM)) = O(N^2 \log(N) + N \log^2(p_{\max}))$, as $N \geq C$ and $N \geq c$. Further, we can assume that $M \leq N$, as for larger M, we would simply distribute all jobs along the machine leaving $M - N$ machines empty. □

8.2 SPLITTABLE CASE

In the splittable case, we are allowed to cut and distribute the jobs arbitrarily between the machines as long as the class restriction is not violated and jobs do not overlap on a machine. The algorithm for solving this problem proceeds similarly to the one for the non-preemptive case: It searches for the optimal makespan via a binary search. In each iteration, classes with $P_u > T$ are divided into $\lceil P_u / T \rceil$ new, unique classes, each with an accumulated processing time smaller or equal to T. Grouping the classes is easy in this case, as we simply cut each class into pieces of size T until the last fraction with load smaller or equal to T remains. Denote by C' the number of current classes. The algorithm checks whether $C' > c \cdot M$. If so, discard the guess. Otherwise, the current guess is lowered until we find the lowest feasible guess T satisfying this condition. Then, all current classes are distributed among the machines via *round robin*. Finally, the original classes are reinserted.

Although, we have to proceed with the binary search more carefully here: First of all, we have to adapt our lower bound to just consider the area argument, i. e., $LB = \sum_{j=1}^{N} p_j / M$, as the largest job can now be split and thus, distributed evenly as well. Further, it is not sufficient to only test the integral guesses produced by the binary search, as the optimal makespan can be fractional. However, omitting the rounding in the binary search would need too many steps, or we have to be satisfied with a finite precision. Fortunately, we can circumvent this obstacle by formulating our binary search more careful. Recall that we divide classes with $P_u > T$ into some new classes, each with $P_{u'} \leq T$. Regarding the algorithm – whose correctness is proven below – the only obstacle preventing us from computing a feasible schedule is the amount of different classes after grouping the large ones. Thus, the only interesting guesses on the makespan are those values where by guessing below, the number of new classes increases. In the following, we call them *borders*. We only have to search along these borders to find the optimal makespan. We do so in the following way: For a class u, the borders are determined by P_u / k for $k \in \{1, \dots M - 1\}$; find the smallest feasible guess via a binary search along these borders for each class $u \in [C]$ separately (but test the feasibility regarding the number of all (grouped and small) classes); output the smallest feasible border of all classes. The following lemma proves the correctness of this adapted binary search:

Lemma 43. *The binary search explained above computes a value smaller or equal to the optimal makespan, needing at most $O(C\log(M))$ iterations.*

Proof. The lower bound $LB = \sum_{j=1}^{N} p_j / M$ corresponds to an equal distribution of processing times onto the machines, which clearly bounds the optimal makespan from beneath. Further, as each machine admits exactly c class slots, c times the maximal accumulated processing time, i. e., $UB = c * \max_u\{P_u\}$, still states a correct upper bound.

Further, it is sufficient to only consider the borders, as solely the number of produced classes prevents a guess from being successful. Further, per definition, this number does not change between two neighboring borders.

Obviously P_u / k for $k \in \{1, \dots M-1\}$ for each class u corresponds to the borders regarding that class, i. e., where guessing below, the class u is split into one additional new classes. Note that we do not have to consider larger values for k, as those would include that we already produced M new classes with load T. Thus, the total area of our current guess would be exceeded contradicting the lower bound.

Proceeding with a binary search along the borders of all classes results in the correct lowest value. First, the number of new classes increases monotonically when lowering the guesses. Secondly, when testing feasibility of some border, we count all (grouped and small) classes. So this corresponds to the smallest number resulting in at most $c \cdot M$ classes overall.

Regarding the running time, we have to compute at most $\log(M)$ bounds which are visited while doing the binary search for each class. Considering all classes separately thus needs $O(C\log(M))$ iterations. □

Having this at hand, we can prove the correctness and the desired ratio of our adapted approximation algorithm. First, we investigate the case that $M \leq N$. The other case is considered afterwards.

Theorem 44. *The approximation algorithm with the above minor adaptions produces a solution σ with $\mu(\sigma) \leq 2 \cdot \mu(\text{OPT}(I))$ for the splittable case in time $O(N^2\log(N))$ if $M \leq N$.*

Proof. Using the adapted binary search from Lemma 43 yields the lowest guess on the makespan such that we get at most $c \cdot M$ classes overall. In each iteration with guess T, we divide each class with $P_u > T$ into $C_u = \lceil P_u / T \rceil$ new ones simply by cutting it into pieces of size T. This number of machines (i. e., class slots) is necessary in any optimal schedule with respect to T to distribute the total load of that class. Again, using round robin and reassigning the original classes via a simple mapping clearly does not exceed the number of available class slots on any machine. For a more detailed explanation on this, see Theorem 42.

We argued that we obtain a feasible schedule. Further, applying Lemma 41, we get that the makespan is at most $\sum_{j=1}^{N} p_j / M + \max\{P_{u'} \mid u' \in [cM]\} \leq LB + T \leq T + T = 2T$, as the optimal makespan has to be at least the lower bound and the largest $P_{u'}$ in the manipulated instance is also bounded by T.

Regarding the running time, we first compute all P_u and the upper and lower bounds in time $O(N)$, as we have to add the processing time of each job

exactly once. By Lemma 43, the adapted binary search admits $O(C\log(M))$ many iterations. For each guess T, the grouping takes time $O(N)$, as we have to go trough each class and thus, each job at most once. Then, we sort all classes in time $O(cM\log(cM))$ and allot them in time $O(cM)$ by round robin. Lastly, we reassign the correct classes via a simple mapping to the corresponding jobs, again in time $O(cM)$. This results in an overall running time of

$$\begin{aligned}&O(N)+O(C\log(M))\cdot O(N)+O(N)+O(cM+cM\log(cM))\\&=O(N)+O(CN\log(M))+O(cM\log(cM))\\&=O(N^2\log(M))+O(NM\log(NM)).\end{aligned}$$

as $N \geq C$ and $N \geq c$. Further, we assume for this case that $M \leq N$ yielding the desired running time of $O(N^2\log(N))$. □

Theorem 45. *The approximation algorithm with the above minor adaptions produces a solution σ with $\mu(\sigma) \leq 2 \cdot \mu(\text{OPT}(I))$ for the splittable case in time $O(N^2\log(M))$ if $M > N$.*

Proof. A large number of machines is first problematic when we sort all classes. It may appear that by introducing the new ones, we end up with $O(cM)$ many. However, most of them have size T due to the slicing of the original jobs. At most C many classes can have a size smaller than T, one for each original class. Thus, instead of saving and sorting all of them, we only do so for the C many with $P_{u'} < T$. For the remaining ones we just store their number. Now, by applying round robin, we just distribute the C classes onto $C \leq N < M$ machines and put an arbitrary class with $P_{u'} = T$ on top. For the remaining ones we just save the number of machines which are filled with two classes of size T. The space has to be sufficient, as we are only allowed to place two classes with $P_{u'} = T$ on top of each other or T is a wrong guess. Hence, the algorithm and the output only use C classes and machine configurations explicitly, while M only appears logarithmically. Thus this algorithm is polynomial with respect to the encoding length, i. e.,

$$\begin{aligned}&O(N)+O(C\log(M))\cdot O(N)+O(N)+O(C\log(C)+\log(M))\\&=O(N^2\log(M)).\end{aligned}$$

This together with the correctness proof and estimation on the quality from Theorem 44 completes the proof. □

8.3 PREEMPTIVE CASE

In the preemptive case, we are again allowed to cut the jobs and place them onto different machines. However, we have to take care that no job is scheduled in parallel in this case, i. e., at the same time on different machines. To handle this, we adapt the algorithm for the splittable case. First, the lower bound LB also has to guarantee that $T \geq p_{\max}$ such that there is enough space to schedule each job sequentially (on different machines). Thus, we

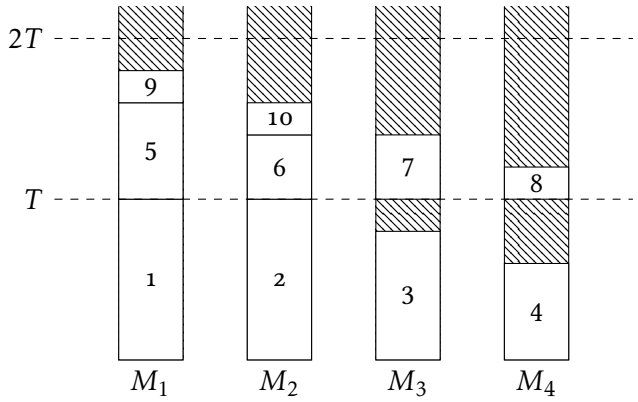

Figure 8.2: The figure shows the repacking for the preemptive case regarding the schedule from Figure 8.1. The approach is to shift the classes above the first one such that they start at T.

compute the lower bound as $LB = \max\{p_{\max}, \sum_{j=1}^{N} p_j / M\}$. Secondly, we have to reschedule some jobs to make sure that no job piece is scheduled in parallel. Note that this only happens while grouping large classes if a job is cut at T and the remaining piece is of size $P_{u'} < T$. Otherwise, each new class with $P_{u'} = T$ is assigned by round robin onto the bottom of a machine due to the lower bound and thus, it does not collide with its other part being at the top. Hence, we only have to repack the machines if there is at least one new class with $P_{u'} = T$. The additional steps to repack the jobs is presented below and is executed after (*) in Algorithm 1. Further, an example showing such a repacking is visualized in Figure 8.2.

If there exists a new class u' with $P_{u'} = T$:
    For each machine: Shift the schedule above the first class such that it starts at time T.

Algorithm 2: Extension of the Algorithm 1 for solving the preemptive case of the CCS problem. It has to be executed after (*).

Theorem 46. *The approximation algorithm with the above minor adaptions produces a solution σ with $\mu(\sigma) \leq 2 \cdot \mu(\text{OPT}(I))$ for the preemptive case in time $O(N^2 \log(N))$.*

Proof. Since it is already proven that the algorithm for the splittable case produces a feasible solution bounded by $2 \cdot \mu(\text{OPT}(I))$, we now focus on the adaptions. The lower bound is computed as $LB = \max\{p_{\max}, \sum_{j=1}^{N} p_j / M\}$, as apart from an equal distribution, the makespan has also to be at least as large as the largest processing time since we cannot process a job in parallel. This also implies that each job is cut by the algorithm at most once while introducing the new classes.

The repacking guarantees that job pieces of the same job are not executed in parallel. As explained above, a job is only cut once at the height of T if a class u has $P_u > T$. Otherwise, we do not have to repack the machines and the estimation from the splittable case holds. Thus, assume we have a class u with $P_u > T$, where by dividing it into C_u new classes, we cut one job. Due to our algorithm, the remaining job piece is either placed at the bottom of a machine, not intersecting with its other part at the top due to our definition of the lower bound, or the job is scheduled above T, also not intersecting.

It remains to prove that, while repacking, we do not exceed a makespan of $2T$ by shifting jobs, which then start and thus, end later. Suppose the contrary, we have a machine M_k with a load $L_k > 2T$. Hence, the classes have to have been shifted and thus, the largest class on that machine has to have a processing time smaller than T. Further, the shifted jobs have a total processing time larger than T to exceed $2T$ by starting at time T. As the first class on M_k has a processing time smaller than T, there exists some other machine M_ℓ with $\ell < k$ admitting a class with $P_{u'} = T$. Thus, the load on M_ℓ excluding the first class with processing time T is also greater than T, as they have to be larger than the load on M_k without the first class, see Lemma 41. Thus, we exceeded $2T$ on a machine without any gaps which is not possible by Lemma 41.

Regarding the running time, it is not reasonable to have more machines than jobs, as we have a lower bound of at least $p_{\max}$. Thus, each job could simply be scheduled separately onto the first N machines obtaining an optimal schedule leaving $M - N$ machines empty. Hence, we assume $N \leq M$ which yields a running time of:

$$\begin{aligned} &O(N) + O(C\log(M)) \cdot O(N) + O(NM\log(NM)) \\ &= O(N^2\log(N) + O(N^2\log(N)) \\ &= O(N^2\log(N)). \end{aligned}$$

for the algorithm, as the other parts stay the same regarding Theorem 44. □

Perhaps there's another, much larger story behind the printed one, a story that changes just as our own world does. And the letters on the page tell us only as much as we'd see peering through a keyhole.

— from *Inkheart* by Cornelia Funke

9

VARIABLE CLASS CONSTRAINT SCHEDULING: PTAS

We already introduced Class Constraint Scheduling in the last chapter. Here, we look at a more general case: Considering the aforementioned scenarios like scheduling on single or multiprocessor computers, distributed manufacturing, or data placement, the locations (e. g.machines) are oftentimes not identical. Instead, machines have different capacities for storing databases or installing operational tools needed for executing the jobs. Thus, we extend the Class Constraint Scheduling problem by adding machine dependent class restrictions. We call this problem Variable Class Constraint Scheduling (VCCS). Again, we aim to find a schedule σ that minimizes the makespan. Along the way, we consider each case of feasible job placement, i. e., the non-preemptive, the splittable and the preemptive case. For formal definitions on the schedule of each case, we refer to Chapter 2.

Formally, we are given a set $\mathcal{J}$ of N jobs, where each job $j \in \mathcal{J}$ has a processing time $p_j \in \mathbb{N}$ and some class $c_j \in \{1, 2, \dots, C\}$. These jobs have to be allotted onto a set $\mathcal{M}$ of M machines, each machine $i \in \mathcal{M}$ with a machine dependent limitation of k_i *class slots*. Recall that the class slots bound the number of different classes a machine can schedule. Assume that $C \leq N$, as otherwise, classes without any job can be discarded. Further, assume $k_i \leq C$. Otherwise, we allow more classes to be scheduled on machine i than classes exist overall. Hence, it is equivalent to reduce larger values of class slots to be at most C.

Denote by M_u the number of machines i with $k_i = u$ class slots for $u = 1, \dots, C$. In the following, we assume that we only have parameter many *machine types*, i. e., we only have ν many different $M_u \neq 0$. The input is described as an instance $I = [p_1, \dots, p_N, c_1, \dots, c_N, M_1, \dots, M_C]$. The encoding length of such an instance is given by

$$\begin{aligned} |I| = O(\sum_{j=1}^{N} \lceil \log(p_j) \rceil + \sum_{j=1}^{N} \lceil \log(c_j) \rceil + \sum_{u=1}^{C} \lceil \log(M_u) \rceil \\ \leq O(N \log(p_{\max}) + N \log(C) + \nu \log(M)) \\ \leq O(N \log(p_{\max}) + N \log(N) + \nu \log(M)). \end{aligned}$$

Denote an optimal schedule with minimum makespan by $\textsc{opt}(I)$. Finding $\textsc{opt}(I)$ is NP-hard for all cases of feasible job placement, as it generalizes the NP-hard Class Constraint Scheduling problem. Thus, we are satisfied to produce a solution close to the makespan of an optimal schedule. In that

matter, we investigate *approximation schemes* that, upon an input ϵ, compute a $(1+\epsilon)$-approximation, i. e., it is guaranteed that we find a schedule σ with $\mu(\sigma) \leq (1+\epsilon) \cdot \mu(\text{OPT}(I))$ arbitrarily close to the makespan of an optimal solution. For details on approximation schemes, we refer to Chapter 2.

In the following, we establish the first polynomial time approximation scheme (PTAS) for each case. That includes the first PTAS for the CCS problem that neither depends on a constant number of classes nor a constant number of class slots.

We do so by using n-fold ILPs. In particular, we show how to model the respective problem in that specific form, which in turn is solvable efficiently. This modeling involves novel insights about optimal solution as well as careful preprocessing and postprocessing steps. By that, we obtain a PTAS for the non-preemptive case with running time $N^{O(\nu^2/\epsilon^{13}\log(1/\epsilon))}O(\log(p_{\max}))$. For the splittable case, we consider a case distinction. The number M only appears logarithmically in the encoding length. As it is reasonable for this case to have more machines than jobs, we thus have to encode the steps and output of the algorithm carefully to obtain a running time polynomial in the encoding length. We do so by proving helpful properties of (nearly) optimal solutions. For the splittable case, this yields algorithms that run in time $N^{O(\nu^2/\epsilon^6\log(1/\epsilon))}\log(p_{\max})$ if $M \leq N$ and in time $N^{O(\nu^2/\epsilon^6\log(1/\epsilon))}\log(M)\log(p_{\max})$ if $M > N$ respectively. Finally, for the preemptive case, we get an algorithm with running time $N^{2^{O(\nu/\epsilon^3)}}\log(p_{\max})$.

Before presenting the PTASs, we elaborate on the techniques we use to obtain them: We assume that there is some accuracy parameter $\delta > 0$ with $1/\delta \in \mathbb{Z}$ depending on ϵ that is specified concretely for each case. Assume that a guess T on the optimal makespan is given. We design a procedure that computes a schedule with makespan $(1+O(\delta))T$ or verifies that a solution with makespan T does not exist. Embedding this in a binary search to find the correct guess on the makespan yields the PTAS. The idea to solve the problem for fixed guesses on the makespan instead of solving the minimization problem directly was introduced by Hochbaum and Shmoys [89].

We call a class $u \in [C]$ *large* if each job of class u has a processing time bigger than $f(\delta)T$ for some computable function f. We call u *small* if there is exactly one job with class u and this job has a processing time of at most $f(\delta)T$. In each of the cases, we first simplify the instance using grouping, merging and rounding techniques. In particular, grouping and merging guarantees that each class u is either small or large. Furthermore, rounding ensures that there are only few distinct processing times.

Then we model our respective problem as an n-fold ILP. This includes proving some novel insights about the structure of (nearly) optimal solutions for the splittable and the preemptive case. We call solutions admitting these properties *well-structured*. We show that if the instance is feasible, then there exists a nearly optimal solution which is well-structured, i. e., there exists a schedule making only a small error where the number of produced job pieces is bounded and further, these pieces are placed at few, specific positions. We

need these insights, as otherwise, we cannot bound the number of variables and thus, cannot set up the desired ILP. Then, by layering the variables hierarchically, it is possible to set up a configuration ILP with the desired block-structure. Finally, we also use the aforementioned structural properties to transform the solution of the ILP into a feasible, nearly optimal schedule.

SUMMARY OF RESULTS

- We establish the first PTAS for the non-preemptive case of the Variable Class Constraint Scheduling problem with running time $N^{O(\nu/\epsilon^{13}\log(1/\epsilon))}O(\log(p_{\max}))$ where ν is a parameter stating the number of machine types.
- We adapt this framework to handle the splittable case of the VCCS problem yielding a PTAS with running time $N^{O(\nu^2/\epsilon^6\log(1/\epsilon))}\log(p_{\max})$ if $M \leq N$. This includes proving adequate structural properties of (nearly) optimal solutions to handle the number of generated job pieces and their positions in the schedule.
- Using structural properties of (nearly) optimal solutions, we manage to encode each step and the output of the algorithm appropriately with respect to the number of machines M to obtain a PTAS for the splittable case with running time $N^{O(\nu^2/\epsilon^6\log(1/\epsilon))}\log(M)\log(p_{\max})$ if $M > N$.
- Finally, we consider the preemptive case. Carefully handling the parallelism and proving structural properties of (nearly) optimal solutions, we manage to obtain a PTAS with running time $N^{2^{O(\nu/\epsilon^3)}}\log(p_{\max})$.

FURTHER RELATED WORK We already discussed the related work for general Scheduling and for the Class Constraint Scheduling problem in Chapter 8. For an overview on algorithms for the n-fold ILPs, we refer to Chapter 3. In the following, we thus focus on the FPT point of view for approximation schemes for Scheduling using block-structured ILPs.

The idea of using n-fold ILPs to construct approximation schemes first appeared in [97]. The authors obtain an n-fold ILP by carefully setting up the configuration ILP for several scheduling problems with setup times, i. e., each job consists of a (machine dependent) processing time and a (machine dependent) setup time.

A preliminary version of these results was published as [100]. In that work, we consider the more restrictive Class Constraint Scheduling where each machine admits the same number of class slots (i. e., $\nu = 1$). The approach is similar. However, the running times presented in this chapter are equally good (considering some minor inaccuracy in the conference version [100]), even though they apply for the more general case.

STRUCTURE OF THIS CHAPTER In Section 9.1, we start with presenting the PTAS for the non-preemptive case of the VCCS problem. Then, Sec-

tion 9.2 presents the PTAS for the splittable case. This includes proving some structural properties of (nearly) optimal solutions regarding the number of produced job pieces and positions in the schedule to bound the variables of the n-fold ILP appropriately. Further, at the end of this section, we show the adaptions to handle the case of $M > N$ machines. Turning our attention to the preemptive case, we show a PTAS in Section 9.3. Again, this includes proving new structural results and integrating these in the n-fold ILP formulation.

9.1 NON-PREEMPTIVE CASE

Let us start with the non-preemptive case, where we have to place the jobs as a whole. Before we can formulate our problem as the desired configuration ILP, we first have to preprocess the instance appropriately. In particular, we aim for an instance where each class either consists of exactly one small job or solely large jobs. Note that a similar approach has been used in [153]. Only then a hierarchical layering of the variables allows us to set up the desired ILP, which in turn can be solved efficiently. Finally, we present how this solution can be used to construct a schedule.

Consider some accuracy parameter $\delta > 0$ with $1/\delta \in \mathbb{Z}$, which we specify in dependence of ϵ later on. Assume some guess T on the makespan.

PREPROCESSING Call a job j *small* if $p_j < \delta^2 T$ and *large* otherwise. First, we construct an instance I' in which each class is either small or large by grouping the jobs. For each class u, we perform the following steps:

- As long as it is possible, repeatedly perform the following steps: Select some set of jobs $X \subseteq \{j \in \mathcal{J} \mid c_j = u, p_j < \delta^2 T\}$ such that the sum of their processing times $p(X)$ lies in the interval $[\delta^2 T, 2\delta^2 T)$; remove X; introduce a new job with class u and size $p(X)$.
- Let $Y = \{j \in \mathcal{J} \mid c_j = u, p_j < \delta^2 T\}$ be the remaining jobs after the first step. We have $p(Y) < \delta^2 T$ because of the above.
- If u contains jobs not belonging to Y, we pick such a job j, remove j and Y from the instance, and introduce a new job of class u with size $p_j + p(Y)$.
- Otherwise, we remove Y from the instance and introduce a new job of class u and with size $p(Y)$.

Note that there are indeed only small and large classes left. We call the resulting instance I', the corresponding set of jobs $\mathcal{J}'$, and write p'_j and c'_j respectively to denote the processing time and class of job $j \in \mathcal{J}'$. The following holds:

Lemma 47. *If there is a schedule with makespan T for I, then there is also a schedule with a makespan of at most $(1+3\delta)T$ for the preprocessed instance I'.*

Proof. We consider some case distinction whether the set Y was grouped to a job j with $p_j \geq \delta T$ or to some job j with $p_j < \delta T$ (this includes that

Y is the only set of jobs in the class). Consider some schedule σ for I with makespan T.

In the first case, we re-schedule the jobs in Y for each class accordingly to the grouping before, i. e., we place them on top of the corresponding large job j. As such a job j has $p_j \geq \delta T$ at most $1/\delta$ many fit on a machine. Each of these jobs produce an error of $p(Y) < \delta^2 T$ by placing a small job on top, resulting in an overall error of at most δT on each machine.

For the other case, we utilize that each grouped job has a size of at most $\delta T + \delta^2 T \leq 2\delta T$. There is a set $\mathcal{J}(j') \subseteq \mathcal{J} \setminus \mathcal{J}'$ for each of these newly introduced jobs $j' \in \mathcal{J}' \setminus \mathcal{J}$ such that $p'_{j'} = \sum_{j \in \mathcal{J}(j')} p_j$ and $\{\mathcal{J}(j') \mid j' \in \mathcal{J}' \setminus \mathcal{J}\}$ is a partition of $\mathcal{J} \setminus \mathcal{J}'$. Given the schedule σ with makespan T, let $y_{i,j} \in \{0,1\}$ be equal to 1 if $j \in \mathcal{J}$ is executed on machine i and equal to 0 otherwise. We set $T_i = \sum_{j \in \mathcal{J} \setminus \mathcal{J}'} y_{ij} p_j$ as the load of the grouped jobs on each machine i. Further, we set $x^*_{ij'} = (\sum_{j \in \mathcal{J}(j')} p_j y_{ij})/p'_{j'}$ for each job $j' \in \mathcal{J}' \setminus \mathcal{J}$ as its fraction on each machine, and $z_{ij'} = \lceil x^*_{ij'} \rceil$ as an indicator stating if the grouped job occurs on the machine. Note that $x^*_{ij'} \in [0,1]$ and $z_{ij'} \in \{0,1\}$. It is easy to see that $(x^*_{ij'})$ is a feasible solution of the following LP:

$$\sum_{j' \in \mathcal{J}' \setminus \mathcal{J}} p'_{j'} x_{ij'} \leq T_i \qquad \forall i \in \mathcal{M} \tag{9.1}$$

$$\sum_{i \in \mathcal{M}} x_{ij'} = 1 \qquad \forall j' \in \mathcal{J}' \setminus \mathcal{J} \tag{9.2}$$

$$0 \leq x_{ij'} \leq z_{ij'} \qquad \forall i \in \mathcal{M}, j' \in \mathcal{J}' \setminus \mathcal{J} \tag{9.3}$$

Employing the classical result by Lenstra et al. [125], we get a rounded solution $(\bar{x}_{ij'})$ such that $\bar{x}_{ij'} \in \{0,1\}$ holds, (9.2) and (9.3) are satisfied, and furthermore, we have $\sum_{j' \in \mathcal{J}' \setminus \mathcal{J}} p'_{j'} \bar{x}_{ij'} \leq T_i + \max_{j' \in \mathcal{J}' \setminus \mathcal{J}} p'_{j'} \leq T_i + 2\delta T$ for each $i \in \mathcal{M}$. Hence, we can generate a suitable schedule by removing the jobs belonging to $\mathcal{J} \setminus \mathcal{J}'$ and assigning the jobs belonging to $\mathcal{J}' \setminus \mathcal{J}$ based on the variables $\bar{x}_{ij'}$ producing an error of at most $2\delta T$ on each machine.

Together, this yields a schedule for I' with an error of at most $\delta T + 2\delta T = 3\delta T$. □

Lastly, we round the processing times as follows: Let j be a job. If c_j is a large class, we set $p''_j = \lceil p'_j/(\delta^3 T) \rceil \delta^3 T$. Clearly, the error is bounded by $\delta^3 T$. If c_j is a small class, we set $p''_j = \lceil p'_j/(\delta^3 T/k_{\max}) \rceil \delta^3 T/k_{\max}$ where $k_{\max}$ is the largest occurring class slot limitation on any machine. This introduces an error of $\delta^3 T$. Overall, this yields a total error by rounding the jobs of $2\delta^3 T$ on each machine. In the following, we call a schedule for the grouped and rounded jobs *well-structured*.

Furthermore, we scale the makespan bound T and the processing times by $k_{\max}/(\delta^3 T)$ to ensure integral values, that is, we have $\delta^3 T/k_{\max} = 1$ afterwards. The resulting instance is denoted by I'', the job set by $\mathcal{J}''$ and the processing time and class of job $j \in \mathcal{J}''$ by p''_j and c''_j respectively. We also write p''_u to denote the processing time of the single, small job of class $u \in [C]$.

Furthermore, we denote the set of rounded processing times occurring in large classes by $\mathcal{P}$ and the number of jobs of class u and size $p \in \mathcal{P}$ by n_p^u.

SETTING UP THE n-FOLD ILP Considering the error produced by grouping and rounding the jobs, we set $\bar{T} = (1+3\delta)(1+2\delta^3)T = (1+O(\delta))T$. Further, we set $\xi_u = 0$ if the corresponding class is large and $\xi_u = 1$ otherwise.

Following the module configuration framework [80] in which *modules* are used to cover the basic objects, we design our n-fold ILP. That is, basic objects correspond to jobs and configurations in turn are used as modules. In this context, we define the set of modules as multiplicity vectors of processing times, i. e., $\Theta = \{\theta \in \mathbb{Z}_{\geq 0}^{\mathcal{P}} \mid \sum_{p\in\mathcal{P}} \theta_p p \leq \bar{T}\}$ summing up to less than the adapted guess $\bar{T}$ on the makespan. In the following ILP, a module is associated with a set of large jobs from a single class. The size $\Lambda(\theta)$ of a module θ is given by $\sum_{p\in\mathcal{P}} \theta_p p$ and the set of module sizes is denoted as $\Lambda(\Theta)$.

Further, we define a configuration $\kappa \in \mathbb{Z}_{\geq 0}^{\Lambda(\Theta)}$ as a multiplicity vector of module *sizes*, and the size $\Lambda(\kappa)$ of a configuration κ is given by $\sum_{q\in\Lambda(\Theta)} \kappa_q q$. The set of feasible configurations $\mathcal{K}_u$ for the machines i with $k_i = u$ is given by the configurations κ with $\Lambda(\kappa) \leq \bar{T}$ and $\|\kappa\|_1 \leq u$. Set $c^* = \min\{\bar{T}/(\delta^2 T), k_{\max}\}$ as the greatest possible number of large classes on a machine. Denote by $\mathcal{K}$ the set of all feasible configurations. Denote by $\mathcal{B}$ the set of possible numbers of remaining class slots after assigning some configuration to some machine. Let $\mathcal{K}(h,b) = \{\kappa \in \mathcal{K} \mid \Lambda(\kappa) = h, \|\kappa\|_1 = b\}$ and $\mathcal{K}_u(h,\bar{b}) = \{\kappa \in \mathcal{K}_u \mid \Lambda(\kappa) = h, u - \|\kappa\|_1 = \bar{b}\}$ for each $h \in \Lambda(\mathcal{K})$, $b \in [c^*]$, and $\bar{b} \in \mathcal{B}$.

Let $u \in [C]$ be a class. We have a variable $y_\theta^u \in \{0,\dots,M\}$ for each module $\theta \in \Theta$ indicating how often θ is chosen to cover the jobs of class u. As we can combine modules for the same class on the same machine to another, larger module, M is an appropriate upper bound on the number of modules for a class. Moreover, we introduce a variable $x_\kappa^u \in \{0,\dots,M\}$ for each configuration $\kappa \in \mathcal{K}$ and machine type u indicating how often the configuration κ is used for machines i with $k_i = u$. Recall that we denote the number of machines with $k_i = u$ by M_u for $u \in [C]$ and that we assume to only have ν different machine types. Thus, some M_u might be 0. Furthermore, for the small classes, we have binary variables $z_{h,\bar{b}}^u \in \{0,1\}$ for each $h \in \Lambda(\mathcal{K})$ and $\bar{b} \in \mathcal{B}$ which are used to decide whether the class is assigned to a machine on which $\bar{b}$ class slots are empty and where an overall size h belonging to large classes are scheduled. The n-fold ILP has the following constraints:

$$\sum_{u=1}^{C} \sum_{\kappa \in \mathcal{K}} x_\kappa^u \kappa_q = \sum_{u=1}^{C} \sum_{\theta \in \Theta : \Lambda(\theta) = q} y_\theta^u \qquad \forall q \in \Lambda(\Theta) \qquad (1)$$

$$\sum_{u=1}^{C} z_{h,\bar{b}}^u \leq \bar{b} \sum_{u=1}^{C} \sum_{\kappa \in \mathcal{K}_u(h,\bar{b})} x_\kappa^u \qquad \forall h \in \Lambda(\mathcal{K}), \bar{b} \in \mathcal{B} \qquad (2)$$

$$\sum_{u=1}^{C} p_u'' z_{h,\bar{b}}^u \leq (\bar{T} - h) \sum_{u=1}^{C} \sum_{\kappa \in \mathcal{K}_u(h,\bar{b})} x_\kappa^u \qquad \forall h \in \Lambda(\mathcal{K}), \bar{b} \in \mathcal{B} \qquad (3)$$

$$\sum_{\theta \in \Theta} \theta_p y_\theta^u = (1 - \xi_u) n_p^u \qquad \forall u \in [C], p \in \mathcal{P} \qquad (4)$$

$$\sum_{h \in \Lambda(\mathcal{K})} \sum_{\bar{b} \in \mathcal{B}} z_{h,\bar{b}}^u = \xi_u \qquad \forall u \in [C] \qquad (5)$$

$$\sum_{\kappa \in \mathcal{K}_u} x_\kappa^u = M_u \qquad \forall u \in [C] \qquad (6)$$

$$\sum_{\kappa \in \mathcal{K}} x_\kappa^u = M_u \qquad \forall u \in [C] \qquad (7)$$

Constraint (1) is satisfied if the module sizes of the chosen configurations cover the chosen modules. Similarly, (4) ensures that the chosen modules indeed cover the large jobs, i. e., the processing time multiplicities introduced by the modules equal the processing times of the jobs. Indirectly, this constraint also ensures for the small classes that these jobs are not covered. Instead, Constraint (5) guarantees that the small job of each class is assigned to exactly one configuration. Simultaneously, the constraints (2) and (3) ensure that the number of empty class slots and the area is sufficient for the small classes. Finally, the constraints (6) and (7) guarantee that we assign exactly M_u configurations to machines with $k_i = u$ for $u \in [C]$ and that these configuration are indeed feasible for the respective machine type.

The constraints (1) through (3) are globally uniform. The remaining constraints are locally uniform and repeated for each class.

We set the cost vector to zero as we are solely aiming for a feasible solution.

Lemma 48. *If there is a schedule with makespan $\bar{T}$ for instance I'', then there is also a solution to the above n-fold ILP.*

Proof. Given a well-structured schedule, each subset of jobs belonging to a large class is included in Θ having a specific size in $\Lambda(\Theta)$. Hence, for each machine, we count for each possible size the number of present module sizes, thereby deriving a configuration. We set the variables x_κ^u accordingly. Let $u \in [C]$. If u is a large class, its jobs may be distributed along the machines, whereas on each machine the jobs of a class build up a module. Thus, we can count the multiplicity of each used module for a certain class and set the variables y_θ^u accordingly and the variables $z_{h,\bar{b}}^u$ to 0. If, on the other hand, u is a small class, then it contains only one job which is scheduled on exactly one machine i. Let κ be the configuration corresponding to i, $h = \Lambda(\kappa)$, and $\bar{b} = k_i - \|\kappa\|_1$. We set $z_{h,\bar{b}}^u = 1$ and $z_{h',\bar{b}'}^u = 0$ for each $(h', \bar{b}') \neq (h, b)$.

Furthermore, we set all the variables y_θ^u to 0. It is easy to verify that this solution is feasible. □

Hence, if the n-fold ILP has no feasible solution, we can reject the makespan guess T.

SOLVING THE n-FOLD ILP We have to bound the parameters corresponding to the block dimensions, the largest occurring coefficient Δ in the constraint matrix and the lower and upper bounds on the variables to apply Proposition 1. Obviously, $w_{\max} = 0$, as we set the cost vector to zero. The sizes of the sets we introduced are estimated as follows:

- $|\mathcal{P}| = O(1/\delta^3)$ by the rounding,
- $|\Theta| = (|\mathcal{P}|)^{1/\delta^2} = 2^{O(1/\delta^2 \log(1/\delta))}$, as we can choose at most $1/\delta^2$ jobs from the present processing times to not exceed the makespan,
- $|\Lambda(\Theta)| = O(1/\delta^3)$ due to the rounding of the large jobs,
- $|\mathcal{K}| = (|\Theta|)^{1/\delta^2} = 2^{O(1/\delta^4 \log(1/\delta))}$, as we can again only have $1/\delta^2$ jobs and thus, modules to not exceed the makespan,
- $|\Lambda(\mathcal{K})| = O(1/\delta^3)$ due to rounding,
- $c^* = O(1/\delta^2)$, as at most $1/\delta^2$ jobs fit on a machine, and
- $|\mathcal{B}| = O(\nu/\delta^2)$ as 0 up to $1/\delta^2$ class slots can be occupied by large jobs leaving $1/\delta^2 + 1$ different values for the remaining class slots for each machine and we have ν machine types with different numbers of initial free class slots.

This yields the following parameters for the n-fold ILP:

- $n = C$,
- $r = |\Lambda(\Theta)| + |\Lambda(\mathcal{K})| \cdot |\mathcal{B}| = O(\nu/\delta^5)$,
- $s = |\mathcal{P}| = O(1/\delta^3)$, and
- $t = |\Lambda(\mathcal{K})| \cdot |\mathcal{B}| + |\mathcal{K}| + |\Theta| = \nu 2^{O(1/\delta^4 \log(1/\delta))}$.

The largest value Δ can be bounded by $\max\{\bar{T}, \bar{b}\} = \max\{k_{\max}/\delta^3, C\} = O(C/\delta^3)$. Applying Proposition 1 yields

$$\begin{aligned}
&2^{O(\nu/\delta^{11})}(C/\delta^{11})^{O(\nu^2/\delta^{13})}(C\nu 2^{O(1/\delta^4 \log(1/\delta))})^{1+o(1)} \\
&= C^{O(\nu^2/\delta^{13} \log(1/\delta))} \\
&= N^{O(\nu^2/\delta^{13} \log(1/\delta))}
\end{aligned}$$

as $C \leq N$ and $\nu \leq C$.

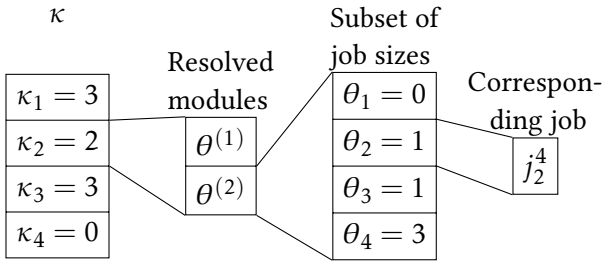

Figure 9.1: This figure partially resolves a configuration κ into a job j_2^4. Here, a configuration consists of occurrences of module sizes $\kappa_1, \dots, \kappa_4$. For example, the second module size appears $\kappa_2 = 2$ times. Meaning, we have a placeholder for two modules of exactly this size, here $\theta^{(1)}$ and $\theta^{(2)}$. Modules themselves hold multiplicities of job sizes, for example module $\theta^{(2)}$ has an occurrence of $\theta_2 = 1$ of the second job size. This space is then filled with the job j_2^4 of class 4 with precisely that size.

CONSTRUCTING THE SCHEDULE Having this solution at hand, we still have to build the schedule. We assign the configurations chosen by the variables x_κ^u onto the machines. Now, we unfold the configurations as Figure 9.1 shows. In detail, given a machine with configuration κ, we create $\|\kappa\|_1$ slots where exactly κ_q slots have size q for each $q \in \Lambda(\Theta)$. Each slot is then greedily filled with a module corresponding to the assignment of y_θ^u. Next, each module θ is resolved into the corresponding multiplicities θ_p of job lengths p. Then, we assign the corresponding jobs greedily into the slots of the job lengths. Afterwards, all large jobs are allotted. Due to the constraints of the ILP, it is easy to see that these steps are successful. Regarding the small jobs, we distribute them greedily as follows: For each $h \in \Lambda(\mathcal{K})$ and $\bar{b} \in \mathcal{B}$, we assign the jobs of the small classes u with $z_{h,\bar{b}}^u = 1$ onto the machines which are filled up to h by large classes and have $\bar{b}$ empty class slots left using round robin. Due to Lemma 41 and (3), this yields a schedule with makespan of at most $\bar{T} + \delta^2 T$. In the last step, we have to reinsert the original, non-rounded processing times and jobs.

The overall running time for placing the large classes onto the machines is linear in the number of involved jobs, i. e., dissolving all configurations takes time $O(M \cdot 1/\delta^2)$. When placing the small jobs, we touch each small job and each class at most once. To insert the original jobs and job sizes, we have to consider each job once. As $M \leq N$ and $C \leq N$, this step yields an overall running time of $O(M \cdot 1/\delta^2 + C + N) = O(N/\delta^2)$.

TOTAL RUNNING TIME AND ERROR We have seen how we can solve the non-preemptive case of the Variable Class Constraint Scheduling problem when we are given a guess T on the makespan. Indeed, a complete algorithm requires to embed the algorithm given above in a binary search. As $N \cdot p_{\max}$ is an upper bound on the largest possible makespan and the optimal

makespan is integral, the search is exhausted after at most $O(\log(N \cdot p_{\max}))$ steps. Thus, we get a total running time of:

$$\underbrace{O(\log(N \cdot p_{\max}))}_{\text{Binary Search}} \cdot [\underbrace{O(N)}_{\text{Preprocess}} + \underbrace{N^{O(\nu^2/\delta^{13}\log(1/\delta))}}_{n\text{-fold ILP}} + \underbrace{O(N/\delta^2)}_{\text{Schedule construction}}]$$
$$= N^{O(\nu^2/\delta^{13}\log(1/\delta))} O(\log(p_{\max})).$$

Since we bounded the error with $O(\delta)$ in each step, the overall error is also at most $O(\delta)$. Setting $\delta = O(\epsilon)$, we get the desired approximation ratio. This completes the algorithm and its analysis. Overall, we get:

Theorem 49. *A schedule σ for the non-preemptive case of the VARIABLE CLASS CONSTRAINED SCHEDULING problem with makespan $\mu(\sigma) \leq (1+\epsilon)\cdot\mu(\text{OPT}(I))$ can be computed in time $N^{O(\nu^2/\epsilon^{13}\log(1/\epsilon))} O(\log(p_{\max}))$, where $\text{OPT}(I)$ denotes a solution with optimal makespan for scheduling the instance I and ν is a parameter stating the number of different machine types. This yields the desired PTAS for this problem.*

9.2 SPLITTABLE CASE

For the most part, the splittable case is quite similar to the non-preemptive one. However, we can assume that each class consists of exactly one job, which simplifies the preprocessing and the definitions of modules and configurations. Nonetheless, there is some extra difficulty arising from the fact that the number of machines can be exponential. If this applies, we manage to lower the dependency on M to logarithmic terms using some insights on the structure and extending our algorithm accordingly. But for now, assume that $M \leq N$. The other case is handled afterwards.

PREPROCESSING In the splittable case, we can simply group all jobs belonging to a class $u \in [C]$ to one job with processing time $p_u = \sum_{j \in \mathcal{J}, c_j = u} p_j$. If we have $p_u > \delta T$, the class u is large. Otherwise, u is small. It is easy to see that the problem is equivalent since the newly created jobs can still be split arbitrarily and behave the same concerning the class constraints.

Next, we round the processing times. Let j be a job. If c_j is a large class, we set its processing time to $p'_j = \lceil p_j/(\delta^2 T)\rceil \delta^2 T$. If not, c_j is a small class and we set $p'_j = \lceil p_j/(\delta^2 T/k_{\max})\rceil \delta^2 T/k_{\max}$. Furthermore, we scale the makespan bound T and the processing times by $k_{\max}/(\delta^2 T)$ to ensure integral values, that is, we have $\delta^2 T/k_{\max} = 1$ afterwards. The resulting instance is denoted by I', the set of jobs by $\mathcal{J}'$ and the processing time and class of job $j \in \mathcal{J}'$ by p'_j and c'_j, respectively. We also write p'_u to denote the processing time of the single job of class $u \in [C]$.

Lemma 50. *If there is a schedule with makespan T for instance I, then there is also a schedule with makespan of at most $(1+2\delta)T$ for I'.*

Proof. A schedule with makespan T for I directly induces a schedule with the same makespan for the instance with the grouped jobs. To realize the

increased processing times, we may distribute the increase proportionally to its job pieces. Note that the jobs belonging to large classes are increased at most by a factor of $(1+\delta)$ by the rounding procedure as each of them had a size of at least δT before. Hence, the load on each machine due to such jobs may increase at most by this factor. Furthermore, there can be at most $k_{\max}$ job pieces belonging to small classes scheduled on each machine, and therefore the increase due to small jobs is upper bounded by $k_{\max} \cdot \delta^2 T / k_{\max} = \delta^2 T$. □

WELL-STRUCTURED SCHEDULE In the splittable case, we call a schedule *well-structured* if the following holds: The size of each split piece of a job belonging to a large class is at least δT and an integer multiple of $\delta^2 T$. Furthermore, jobs belonging to small classes are not split at all.

Lemma 51. *If there is a schedule with makespan T' for instance I', then there is also a well-structured schedule with makespan of at most $T' + 2\delta T$ for I'.*

Proof. Let there be a schedule with makespan T' for instance I' and $j \in \mathcal{J}'$. If c_j is a large class, we divide j into $n_j := \lfloor p'_j / (\delta T) \rfloor$ many parts. The size $s_{j,\ell}$ of the ℓ-th part for each $\ell \in [n_j - 1]$ is δT and we set $s_{j,n_j} = p'_j - \sum_{\ell=1}^{n_j-1} s_{j,\ell} \in [\delta T, 2\delta T)$. Note that all the parts have a size that is an integer multiple of $\delta^2 T$ due to the rounding. The schedule for j translates into a schedule for the job parts in a straight-forward fashion. If c_j is a small class, we have just one job part given by the whole job and therefore, set $n_j = 1$ as well as $s_{j,1} = p'_j$. Now, let $x^*_{(j,\ell),i}$ be the fraction of the ℓ-th part of job j that is assigned to machine i in the given schedule. Furthermore, let $z_{(j,\ell),i} \in \{0,1\}$ be equal to 1 if some piece of the ℓ-th part of job j is assigned to machine i and equal to 0 otherwise. It is easy to verify that $(x^*_{(j,\ell),i})$ is a feasible solution of the following LP:

$$\sum_{j \in \mathcal{J}', \ell \in [n_j]} s_{j,\ell} x_{(j,\ell),i} \leq T' \qquad \forall i \in, \mathcal{M} \tag{9.4}$$

$$\sum_{i \in \mathcal{M}} x_{(j,\ell),i} = 1 \qquad \forall j \in \mathcal{J}', \ell \in [n_j] \tag{9.5}$$

$$0 \leq x_{(j,\ell),i} \leq z_{(j,\ell),i} \qquad \forall i \in \mathcal{M}, j \in \mathcal{J}', \ell \in [n_j] \tag{9.6}$$

Employing a classical rounding result by Lenstra et al. [125] yields a rounded solution $(\bar{x}_{(j,\ell),i})$ such that $\bar{x}_{(j,\ell),i} \in \{0,1\}$ holds, (9.5) and (9.6) are satisfied, and furthermore, we have $\sum_{j \in \mathcal{J}', \ell \in [n_j]} s_{j,\ell} \bar{x}_{(j,\ell),i} \leq T' + \max_{j \in \mathcal{J}', \ell \in [n_j]} s_{j,\ell} \leq T' + 2\delta T$ for each $i \in \mathcal{M}$. The rounded solution directly yields a well-structured schedule for I' with makespan at most $T' + 2\delta T$. □

SETTING UP THE n-FOLD Taking the above steps into consideration, we set $\bar{T} = (1 + O(\delta))T$ and search for a well-structured schedule with makespan $\bar{T}$ via an n-fold ILP. Further, we set $\xi_u = 0$ if the corresponding class is large and $\xi_u = 1$ otherwise.

First, let us define modules and configurations for this case. These resemble the non-preemptive case, but, as we only have one job for each class,

things become a little handier: Define the set of modules Θ to be the set of possible split sizes of jobs from large classes in a well-structured schedule with makespan $\bar{T}$, that is, $\Theta = \left\{\ell\delta^2 T \,\middle|\, \ell \in \{1/\delta, \dots, \bar{T}/(\delta^2 T)\}\right\}$. Note that a module indirectly states its size.

A configuration $\kappa \in \mathbb{Z}_{\geq 0}^{\Theta}$ is a multiplicity vector of modules (module sizes) and its size $\Lambda(\kappa)$ is given by $\sum_{q\in\Theta} \kappa_q q$. Intuitively, each module in a configuration should belong to a distinct class, as otherwise, two modules could be combined to one. Thus, $\|\kappa\|_1 = \sum_{q\in\Theta} \kappa_q$ corresponds to the number of class slots used in the configuration. We consider the set $\mathcal{K}$ of feasible configurations κ with $\sum_{q\in\Theta} \kappa_q q \leq \bar{T}$ and $\|\kappa\|_1 \leq k_{\max}$ and denote the set of feasible configurations for a machine type u by $\mathcal{K}_u$. The set of configuration sizes is $\Lambda(\mathcal{K})$.

Let $\kappa \in \mathcal{K}$. Since $q \geq \delta T$ for each $q \in \Theta$, we know that $\|\kappa\|_1 \leq \bar{T}/\delta T = O(1/\delta)$ is the greatest possible number of large classes on a machine. We set $c^* = \min\{\bar{T}/\delta T, k_{\max}\}$. For each $h \in \Lambda(\mathcal{K})$ and $b \in [c^*]$, we define $\mathcal{K}(h,b) = \{\kappa \in \mathcal{K} \,|\, \Lambda(K) = h, \|K\|_1 = b\}$. Again, denote by $\mathcal{B}$ the set of possible numbers for remaining class slots on any machine after placing some configuration. Let $\mathcal{K}_u(h,\bar{b}) = \{\kappa \in \mathcal{K}_u \,|\, \Lambda(\kappa) = h, u - \|\kappa\|_1 = \bar{b}\}$ for each $h \in \Lambda(\mathcal{K})$ and $\bar{b} \in \mathcal{B}$.

We again introduce three types of variables. We have a variable $y_q^u \in \{0,\dots,M\}$ for each module $q \in \Theta$ indicating how often q is chosen to cover the job of class u. Moreover, we introduce a variable $x_\kappa^u \in \{0,\dots,M\}$ for each configuration $\kappa \in \mathcal{K}$ and machine type u indicating how often the configuration κ is used for machines i with $k_i = u$. We use the above variables to handle the assignment of large classes. To deal with the small classes, we have binary variables $z_{h,\bar{b}}^u \in \{0,1\}$ for each $h \in \Lambda(\mathcal{K})$ and $\bar{b} \in \mathcal{B}$ which are used to decide whether the class is assigned to a machine on which $\bar{b}$ class slots are empty and where an overall size h belonging to large classes are scheduled. The actual schedule of the small classes is again determined using the round robin procedure for each size h and number of remaining empty class slots $\bar{b}$. Recall that we denote by M_u the number of machines i with $k_i = u$. Further, we assume that only ν machine types are non-empty. The n-fold has the following constraints:

$$\sum_{u=1}^{C} \sum_{\kappa \in \mathcal{K}} x_\kappa^u \kappa_q = \sum_{u=1}^{C} y_q^u \qquad \forall q \in \Theta \qquad (1)$$

$$\sum_{u=1}^{C} z_{h,\bar{b}}^u \leq \bar{b} \sum_{u=1}^{C} \sum_{\kappa \in \mathcal{K}_u(h,\bar{b})} x_\kappa^u \qquad \forall h \in \Lambda(\mathcal{K}), \bar{b} \in \mathcal{B} \qquad (2)$$

$$\sum_{u=1}^{C} p'_u z_{h,\bar{b}}^u \leq (\bar{T} - h) \sum_{u=1}^{C} \sum_{\kappa \in \mathcal{K}_u(h,\bar{b})} x_\kappa^u \qquad \forall h \in \Lambda(\mathcal{K}), \bar{b} \in \mathcal{B} \qquad (3)$$

$$\sum_{q \in \Theta} q y_q^u = (1 - \xi_u) p'_u \qquad \forall u \in [C] \qquad (4)$$

$$\sum_{h \in \Lambda(\mathcal{K})} \sum_{\bar{b} \in \mathcal{B}} z_{h,\bar{b}}^u = \xi_u \qquad \forall u \in [C] \qquad (5)$$

$$\sum_{\kappa \in \mathcal{K}_u} x_\kappa^u = M_u \qquad \forall u \in [C] \qquad (6)$$

$$\sum_{\kappa \in \mathcal{K}} x_\kappa^u = M_u \qquad \forall u \in [C] \qquad (7)$$

The constraints carry the same meaning as for the previous n-fold ILP. In detail, Constraint (1) is satisfied if the chosen configurations cover the chosen modules; and due to Constraint (4) the chosen modules cover the job of a class if that class is large. If a class is small, the constraints (4) and (5) ensure that no modules are chosen for this class and that the job of this class is assigned to exactly one type of configuration, respectively. The constraints (2) and (3) ensure that there is a proper amount of space and class slots for the small classes left. The remaining ones, i. e., the constraints (6) and (7) handle the assignment of machines to configurations, i. e., we chose the correct number of feasible configuration for each machine type.

Clearly, the last four constraints are locally uniform (repeated over the classes) and the remaining ones are globally uniform.

Again, we set the cost vector to zero.

Lemma 52. *If there is a well-structured schedule with makespan $\bar{T}$ for instance I', then there is also a solution to the above n-fold ILP.*

Proof. Given a well-structured schedule, the size of each job piece belonging to a large class scheduled on any machine is included in Θ. Hence, for each machine we may count for each possible size the number of present pieces and thereby, derive a configuration. We set the variables x_κ^u accordingly. If u is a large class, it is split into pieces with sizes included in Θ for the schedule. We set the variables y_q^u accordingly and the variables $z_{h,\bar{b}}^u$ to 0. If u is a small class, the whole class is scheduled on the same machine i. Let κ be the configuration corresponding to i, $h = \Lambda(K)$, and $\bar{b} = k_i - \|\kappa\|_1$. We set $z_{h,\bar{b}}^u = 1$ and $z_{h',b'}^u = 0$ for each $(h', b') \neq (h, b)$. Furthermore, we set all variables y_q^u to 0. It is easy to verify that this solution is feasible. □

Hence, if the n-fold has no feasible solution, we can reject the makespan guess T.

SOLVING THE n-FOLD Before applying Proposition 1, we have to estimate the size of the parameters. First, let us bound the magnitude of the generated sets and numbers:

- $|\mathcal{P}| = O(1/\delta^2)$ by to the rounding,
- $|\Theta| = O(1/\delta^2)$ due to the definition of modules,
- $|\mathcal{K}| = O(|\Theta|^{1/\delta}) = 2^{O(1/\delta \log(1/\delta))}$, as we can have at most $1/\delta$ job pieces and thus modules to not exceed the makespan,
- $|\Lambda(\mathcal{K})| = O(1/\delta^2)$ due to the size and number of possible job pieces,
- $c^* = O(1/\delta)$, as at most $1/\delta$ jobs fit on a machine, and
- $|\mathcal{B}| = O(\nu/\delta)$, as 0 up to $1/\delta$ class slots can be occupied by large jobs (job pieces) leaving $1/\delta + 1$ different values for the remaining class slots for each machine and we have ν machine types with different numbers of initial free class slots.

This yields the following dimensions of the n-fold ILP:

- $n = C$,
- $r = |\Theta| + |\Lambda(\mathcal{K})| \cdot |\mathcal{B}| = O(\nu/\delta^3)$,
- $s = O(1)$, and
- $t = |\mathcal{K}| + |\Theta| + |\Lambda(\mathcal{K})| \cdot |\mathcal{B}| = \nu 2^{O(1/\delta \log(1/\delta))}$.

The largest absolute value of each number in the constraint matrix is upper bounded by $\bar{T}$. Due to the scaling and as $k_{\max} \leq C$ we have $\bar{T} = O(C/\delta^2)$. Applying Proposition 1 and using that $C \leq N$ gives us a running time to solve the n-fold ILP of:

$$2^{O(\nu/\delta^3)}(C/\delta^5)^{O(\nu^2/\delta^6)}(C\nu 2^{O(1/\delta \log(1/\delta))})^{1+o(1)}$$
$$= C^{O(\nu^2/\delta^6 \log(1/\delta))} = N^{O(\nu^2/\delta^6 \log(1/\delta))}.$$

CONSTRUCTING THE SCHEDULE Given a solution, we still have to build the schedule. For each large class u, we split the job of class u into y^u_q pieces of size q for each $q \in \Theta$. Next, we assign the configurations chosen by the variables x^u_κ onto the machines. Given a machine with configuration κ, we create κ_q slots of size q for each $q \in \Theta$. Then, we assign the job pieces greedily into fitting placeholders on the machine. It is easy to see that these steps are successful due to the constraints of the n-fold ILP. Lastly, we have to assign the small classes. To do so, we again employ the round robin approach: For each $h \in \Lambda(\mathcal{K})$ and $\bar{b} \in \mathcal{B}$, we assign the jobs of the small classes u with $z^u_{h,\bar{b}} = 1$ onto the machines where h space is occupied by large classes and $\bar{b}$ class slots are left via round robin. Due to (2), all the jobs can be placed by this procedure. Furthermore, due to Lemma 41 and (3), this yields a schedule with makespan of at most $\bar{T} + \delta T$. Lastly, we have to use the original running times and jobs, which can be done using a greedy approach.

The overall running time for placing the large classes is linear in the number of involved job pieces, that is $O(M/\delta)$. When placing the small jobs, we touch each class at most once. Further, when we insert the original jobs and job sizes, we have to consider each job and job piece in the schedule once. The overall running time can thus be bounded by $O(M/\delta + C) = O(N/\delta)$ as $C \leq N$ and $M \leq N$ holds for this case.

TOTAL RUNNING TIME AND ERROR We have seen how to solve the splittable case when we are given a guess T on the makespan. So, we embed this algorithm in a binary search for the optimal makespan. This can be done in $O(\log(N \cdot p_{\max}/\delta))$ iterations as $N \cdot p_{\max}$ is an upper bound on the largest possible makespan and we are satisfied with a precision of $O(\delta)$. Using $C \leq N$, we get a total running time of:

$$\underbrace{O(\log(\frac{N \cdot p_{\max}}{\delta}))}_{\text{Binary Search}} \cdot [\ \underbrace{O(N)}_{\text{Preprocess}} + \underbrace{N^{O(\nu^2/\delta^6 \log(1/\delta))}}_{n\text{-fold ILP}} + \underbrace{O(N/\delta)}_{\text{Schedule construction}}\ ]$$
$$= N^{O(\nu^2/\delta^6 \log(1/\delta))} \log(p_{\max}).$$

Furthermore, the error in every phase can be bounded by $O(\delta)$ as we analyzed above. Thus, the overall error is given by $O(\delta)$. Setting $\delta = O(\epsilon)$, we get the desired approximation ratio. This yields the total algorithm and analysis for the problem. The next theorem summarizes the results.

Theorem 53. *A schedule σ for the splittable case of the VARIABLE CLASS CONSTRAINED SCHEDULING problem with makespan $\mu(\sigma) \leq (1+\epsilon) \cdot \mu(\text{OPT}(I))$ can be computed in running time $N^{O(\nu^2/\epsilon^6 \log(1/\epsilon))} \log(p_{\max})$ if $M \leq N$, where $\text{OPT}(I)$ denotes a solution with optimal makespan for scheduling the instance I. Further, ν is a parameter stating the number of machine types. This yields the desired PTAS for this problem.*

HANDLING AN EXPONENTIAL NUMBER OF MACHINES In this case, it is reasonable to have (arbitrarily many) more machines than jobs. In that case, the algorithm described above would not compute a solution in polynomial time regarding the encoding length of the instance. However, we can handle this by extending our algorithm using an idea from [97].

First, observe that we can convert any schedule into a schedule in which each machine has the same load and each pair of classes occurs on at most one machine. Indeed, if we have two machines i_1 and i_2 on which the same pair (u_1, u_2) occurs, we can apply a simple swap: Let $p(i,u)$ be the overall load of a class $u \in \{u_1, u_2\}$ on a machine $i \in \{i_1, i_2\}$. W. l. o. g., we may assume that $p(i_1, u_1)$ is minimal. We move all the job pieces of class u_1 placed on machine i_1 to machine i_2 and job pieces of class u_2 with overall size $p(i_1, u_1)$ from machine i_2 to i_1. Afterwards, both machines have the same load, and class u_1 does not occur on machine i_1. Moreover, the number of used class slots has not increased on any machine. The approach is visualized in Figure 9.2. In a second step, we can transform the schedule even further.

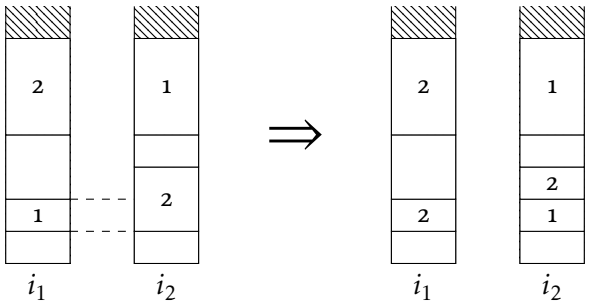

Figure 9.2: This figure shows the exchange of two job pieces, such that the machines have distinct pairs of classes.

For each class u, we can guarantee that there is at most one machine that exclusively executes pieces belonging to u and is not fully filled. Again, this is realized via a simple swapping argument.

We can modify the n-fold ILP correspondingly: There are two configurations that we call trivial, namely the one that chooses the largest module exactly once, and the one which does not choose any module. Let $\mathcal{K}'$ be the subset of non-trivial configurations of $\mathcal{K}$. Due to the above considerations, we may introduce the following globally uniform constraint without violating Lemma 52:

$$\sum_{u=1}^{C} \sum_{\kappa \in \mathcal{K}'} x_\kappa^u \leq \binom{C}{2} + C$$

It is easy to verify that the increase in the running time vanishes in the big O notation.

Now, when constructing the schedule, we first deal with the trivial configurations and remember for each class the corresponding number of machines that are fully filled with these classes. The sizes of the classes are decreased accordingly and in the following only at most $\binom{C}{2} + C$ machines are taken into account. Besides this, the algorithm stays the same. Using this approach yields the theorem:

Theorem 54. *A schedule σ for the splittable case of the* Variable Class Constrained Scheduling *problem with makespan $\mu(\sigma) \leq (1+\epsilon)\cdot\mu(\mathrm{OPT}(I))$ can be computed in running time $N^{O(v^2/\epsilon^6 \log(1/\epsilon))} \log(M) \log(p_{\max})$ if $M > N$, where $\mathrm{OPT}(I)$ denotes a solution with optimal makespan for scheduling the instance I. Further, v is a parameter stating the number of machine types. This yields the desired PTAS for this problem.*

9.3 PREEMPTIVE CASE

In the preemptive case, we are allowed to split jobs arbitrarily as long as pieces belonging to the same job are not executed in parallel. This additional constraint makes it the hardest case. However, we handle these obstacles by

proving some nice structures about nearly optimal solutions. In particular, we show that jobs are split evenly and boundedly often, and that these pieces are placed only at few, distinct positions. Using this, we can then formulate the n-fold ILP. Again, adapting the remaining steps of the algorithms presented before, we get the desired PTAS.

PREPROCESSING The grouping equals the one from the non-preemptive case. By that, we derive an instance I' with a set of jobs $\mathcal{J}'$ and the property that each class is either large or small. We have:

Lemma 55. *If there is a schedule with makespan T for I, then there is also a schedule with makespan $(1+3\delta)T$ for instance I' in which each job belonging to a small class is completely scheduled on one machine.*

Proof. The proof resembles the one of Lemma 47. Consider some feasible schedule σ for I with makespan T. Again, we make some case distinction whether the set Y was grouped to a large job j with $p_j \geq \delta T$ or it was grouped to some job j with $p_j < \delta T$ (including the case that it is the only job of the class). We define $y_{i,j} \in [0,1]$ to be the fraction of job $j \in \mathcal{J}$ that is scheduled on machine i.

In the first case, we assign the fraction $y_{i,j}$ for each large job j with $p_j \geq \delta T$ and each machine i of the corresponding set Y onto i. To avoid executing these fractions of Y in parallel, we use the fact that the job pieces of the large job are not scheduled in parallel (σ is feasible). In particular, we copy the schedule of each machine and place it on top, scaled down by a factor of δ. These entries of the schedule serve as placeholders, i. e., we place the fractions of Y into the placeholders of the corresponding large job. The space is sufficient, as Y is at least δ smaller as this job. This produces an error of δT.

For the other case and remaining small jobs, we employ the classical result by Lenstra et al. [125] similar to the non-preemptive case. The main difference is that we defined $y_{i,j} \in [0,1]$ to be the fraction of job $j \in \mathcal{J}$ that is scheduled on machine i. When constructing the schedule for I' from the variables $\bar{x}_{i,j'}$, we place the jobs from $\mathcal{J}' \setminus \mathcal{J}$ on top of each machine. Note that we do not have to change the approach to guarantee that each job belonging to a small class is completely scheduled on one machine afterwards. As the largest (grouped) job considered here has a size of $2\delta T$, we produce an error of that size.

Together, this yields a total error of at most $\delta + 2\delta T = 3\delta T$. □

Further, we round and scale the processing times and the makespan as in the non-preemptive case and call the resulting instance I''. For small classes u, we write p''_u to denote the processing time of the single job of that class. As before, we denote the set of rounded processing times occurring in large classes by $\mathcal{P}$ and the number of jobs of class u and size $p \in \mathcal{P}$ by n^u_p.

WELL-STRUCTURED SCHEDULE In the preemptive case, we call a schedule *well-structured* if the following two conditions hold:

- Each job belonging to a small class is completely scheduled on one machine.
- For jobs belonging to large classes, each job piece starts at a multiple of $\delta^3 T$ and its size is a multiple of $\delta^3 T$.

Given some fixed makespan bound T', we define the set of layers L as $\{\ell \in \mathbb{Z}_{>0} | (\ell-1)\delta^3 T \leq T'\}$. If a job j is at least partially scheduled in the interval $[(\ell-1)\delta^3 T, \ell\delta^3 T)$ for some $\ell \in \mathbb{Z}_{>0}$, we say that job j is placed in layer ℓ. Moreover, we call pairs of layers and machines *windows*. Denote the set of windows as $W = \mathcal{M} \times L$. We say that a job j is placed in a window $w = (i, \ell)$ if it is (partially) scheduled on i in layer ℓ. Obviously, in a well-structured schedule, pieces of jobs belonging to large classes that are placed in some window have to fill the window completely.

Lemma 56. *If there is a schedule with makespan T' for instance I' in which each job belonging to a small class is completely scheduled on one machine, then there is also a well-structured schedule for I' with makespan of at most $T' + \delta^3 T$.*

Proof. Let there be a schedule with makespan T' for instance I' in which each job belonging to a small class is completely scheduled on one machine. For each machine i, let D_i denote the overall processing time of jobs belonging to large classes that is executed on i. Moreover, let $\hat{\mathcal{J}}$ denote the set of jobs belonging to large classes, and for each job j and machine i, let $\chi_{i,j} = 1$ if a job belonging to the same class as j is scheduled on i. Otherwise, set $\chi_{i,j} = 0$.

We construct a flow network with the following nodes:

- A source α and a sink ω,
- x_j for each job $j \in \hat{\mathcal{J}}$,
- $u_{j\times\ell}$ for each job $j \in \hat{\mathcal{J}}$ and layer $\ell \in L$,
- $v_{i\times\ell}$ for each window $(i, \ell) \in W$,
- and y_i for each machine $i \in \mathcal{M}$.

Furthermore, we have the following edges and capacities:

- (α, x_j) for each $j \in \hat{\mathcal{J}}$ with capacity $p_j/(\delta^3 T)$ (Note that p_j is a multiplicity of $\delta^3 T$ due to the rounding and thus, $p_j/(\delta^3 T)$ is integral),
- $(x_j, u_{j\times\ell})$ for each $j \in \hat{\mathcal{J}}$ and $\ell \in L$ with capacity 1,
- $(u_{j\times\ell}, v_{i\times\ell})$ for each $j \in \hat{\mathcal{J}}$, $i \in \mathcal{M}$ and $\ell \in L$ with capacity $\chi_{i,j}$,
- $(v_{i\times\ell}, y_i)$ for each $i \in \mathcal{M}$ and $\ell \in L$ with capacity 1,
- and (y_i, ω) for each $i \in \mathcal{M}$ with capacity $\lceil D_i/(\delta^3 T)\rceil$.

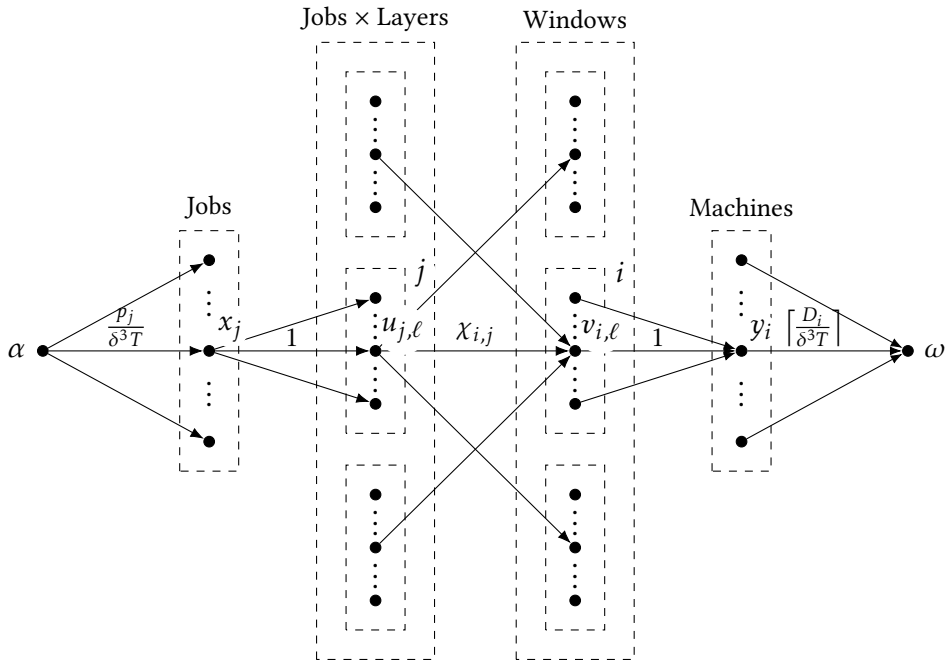

Figure 9.3: The flow network used to prove the existence of a well-structured schedule. Only edges incident to the nodes on the middle vertical axes (α, x_j, $u_{j,\ell}, \dots$) are added.

The construction is summarized in Figure 9.3.

Note that all the capacities are integral and $\sum_{j \in \hat{\mathcal{J}}} p_j/(\delta^3 T)$ is an obvious upper bound for a maximum flow in the network. Let $p(i,j,\ell)$ be the processing time of job j placed in window (i,ℓ) in the given schedule. It is not hard to verify that we get a feasible flow f with value $\sum_{j \in \hat{\mathcal{J}}} p_j/(\delta^3 T)$ by setting:

η	(α, x_j)	$(x_j, u_{j\times\ell})$	$(u_{j\times\ell}, v_{i\times\ell})$	$(v_{i\times\ell}, y_i)$	(y_i, ω)
$f(\eta)$	$\frac{p_j}{\delta^3 T}$	$\sum_{i \in \mathcal{M}} \frac{p(i,j,\ell)}{\delta^3 T}$	$\frac{p(i,j,\ell)}{\delta^3 T}$	$\sum_{j \in \hat{J}} \frac{p(i,j,\ell)}{\delta^3 T}$	$\frac{D_i}{\delta^3 T}$

Thus, by flow integrality, there also exists an integral flow $\bar{f}$ with value $\sum_{j \in \hat{\mathcal{J}}} p_j/(\delta^3 T)$. We use $\bar{f}$ to define a schedule for the jobs in $\hat{\mathcal{J}}$. We do so by defining the processing time $\bar{p}(i,j,\ell)$ of job j placed in window (i,ℓ) in the new schedule. Note that $\bar{f}(e) \in \{0,1\}$ for any edge e of the second, third or fourth type. We set $\bar{p}(i,j,\ell) = \bar{f}(u_{j\times\ell}, v_{i\times\ell})\delta^3 T$. Due to the structure of the flow network, we have:

- For each job j and layer ℓ, there is at most one machine i such that $\bar{p}(i,j,\ell) > 0$.
- For each machine i and layer ℓ, there is at most one job j such that $\bar{p}(i,j,\ell) > 0$.
- For each job j, we have $p_j/(\delta^3 T) = \sum_{i \in \mathcal{M}} \sum_{\ell \in L} \bar{p}(i,j,\ell)$.
- For each machine i, we have $\sum_{j \in \hat{\mathcal{J}}} \sum_{\ell \in L} \bar{p}(i,j,\ell) \leq D_i + \delta^3 T$.

Hence, if we schedule the jobs belonging to small classes on the same machines as before and place them greedily into the gaps, we get a feasible schedule with makespan at most $T' + \delta^3 T$. □

SETTING UP THE n-FOLD Taking the above steps into considerations, we set $\bar{T} = (1 + O(\delta))T$ and search for a well-structured schedule with makespan $\bar{T}$ via an n-fold ILP. Set $\xi_u = 0$ if the corresponding class is large and $\xi_u = 1$ otherwise.

We define the set of layers L with respect to this makespan bound, that is, $L = \{\ell \in \mathbb{Z}_{>0} \,|\, (\ell - 1)\delta^3 T \leq \bar{T}\}$. In a well-structured schedule, jobs fill up whole windows on a given machine, and the windows filled up by jobs of a certain class may be distributed in any possible way on that machine. Hence, we define modules in this context as 0-1-vectors indexed by the layers that include at least one 1, i. e., $\Theta = \{0,1\}^L \setminus \{(0,\dots,0)^\top\}$. Moreover, we define feasible configurations for some machine type $u \in [C]$ as 0-1-vectors indexed by the modules such that at most u modules are chosen, and no two modules occupying the same layer are chosen, that is, $\mathcal{K}_u = \left\{\kappa \in \{0,1\}^\Theta \,\middle|\, \|\kappa\|_1 \leq u, \forall \ell \in L : \sum_{\theta\in\Theta} \kappa_\theta \theta_\ell \leq 1\right\}$. Denote by $\mathcal{K}$ the set of all feasible configurations and by $\mathcal{K}_u$ the set of feasible configurations for a machine type u. Recall that M_u is the number of machines i with type u, i. e., $k_i = u$ and we only have ν different non-empty machine types. The size of a configuration $\Lambda(\kappa)$ is determined by the number of filled up windows, i. e., $\Lambda(\kappa) = \delta^3 T \sum_{\theta\in\Theta} \kappa_\theta \|\theta\|_1$. Note that $\|\kappa\|_1 \leq |L| = O(1/\delta^3)$ for each $\kappa \in \mathcal{K}$. Correspondingly, we set $c^* = \min\{k_{\max}, |L|\}$. We define $\mathcal{K}(h,b) = \{\kappa \in \mathcal{K} \,|\, \Lambda(K) = h, \|K\|_1 = b\}$ for each $h \in \Lambda(\mathcal{K})$ and $b \in [c^*]$. Denote by $\mathcal{B}$ the set of possible numbers of remaining class slots after placing some configuration onto some machine. Let $\mathcal{K}_u(h,\bar{b}) = \{\kappa \in \mathcal{K}_u \,|\, \Lambda(\kappa) = h, u - \|\kappa\|_1 = \bar{b}\}$ for each $h \in \Lambda(\mathcal{K})$ and $\bar{b} \in \mathcal{B}$.

Let $u \in [C]$ be a class. We have a variable $y^u_\theta \in \{0,\dots,M\}$ for each module $\theta \in \Theta$ indicating how often θ is chosen to cover the jobs of class u. Moreover, we introduce a variable $x^u_\kappa \in \{0,\dots,M\}$ for each configuration $\kappa \in \mathcal{K}$ and machine type $u \in [C]$ indicating how often the configuration κ is used for machines i with $k_i = u$. Further, we have binary variables $z^u_{h,\bar{b}} \in \{0,1\}$ for each $h \in \Lambda(\mathcal{K})$ and $\bar{b} \in \mathcal{B}$ which are used to decide whether the class is assigned to a machine on which $\bar{b}$ class slots are empty and where an overall size h belonging to large classes are scheduled. Lastly, we introduce variables $a^u_{p,\ell} \in \{0,\dots,M\}$ for each processing time $p \in \mathcal{P}$ and layer ℓ. These variables are used to determine how many windows in a given layer ℓ are filled by jobs with size p belonging to class u. The n-fold ILP has the following constraints:

$$\sum_{u=1}^{C} \sum_{\kappa \in \mathcal{K}} x_\kappa^u K_\theta = \sum_{u=1}^{C} y_\theta^u \qquad \forall \theta \in \Theta \qquad (1)$$

$$\sum_{u=1}^{C} z_{h,\bar{b}}^u \leq \bar{b} \sum_{u=1}^{C} \sum_{\kappa \in \mathcal{K}_u(h,\bar{b})} x_\kappa^u \qquad \forall h \in \Lambda(\mathcal{K}), \bar{b} \in \mathcal{B} \qquad (2)$$

$$\sum_{u=1}^{C} p'_u z_{h,\bar{b}}^u \leq (\bar{T} - h) \sum_{u=1}^{C} \sum_{\kappa \in \mathcal{K}_u(h,\bar{b})} x_\kappa^u \qquad \forall h \in \Lambda(\mathcal{K}), \bar{b} \in \mathcal{B} \qquad (3)$$

$$\sum_{\ell \in L} a_{p,\ell}^u = (1 - \xi_u) \frac{p}{\delta^3 T} n_p^u \qquad \forall u \in [C], p \in \mathcal{P} \qquad (4)$$

$$\sum_{\theta \in \Theta} \theta_\ell y_\theta^u = \sum_{p \in \mathcal{P}} a_{p,\ell}^u \qquad \forall u \in [C], \ell \in L \qquad (5)$$

$$\sum_{h \in \Lambda(\mathcal{K})} \sum_{\bar{b} \in \mathcal{B}} z_{h,\bar{b}}^u = \xi_u \qquad \forall u \in [C] \qquad (6)$$

$$\sum_{\kappa \in \mathcal{K}_u} x_\kappa^u = M_u \qquad \forall u \in [C] \qquad (7)$$

$$\sum_{\kappa \in \mathcal{K}} x_\kappa^u = M_u \qquad \forall u \in [C] \qquad (8)$$

Note that the n-fold ILP is similar to the ones presented so far. The main difference lies in the changed definitions of modules and configurations and in the constraints (4) and (5). Due to Constraint (4), it is guaranteed that a proper number of windows is reserved to place all the jobs of a certain class and size. Further, these windows are properly covered by modules due to Constraint (5). These new constraints are locally uniform.

As before, we set the cost vector to zero.

Lemma 57. *If there is a schedule with makespan $\bar{T}$ for instance I'', then there is also a solution to the above n-fold ILP.*

Proof. Given a well-structured schedule, the windows occupied by job pieces belonging to a large class define the modules included in Θ. The combination of modules appearing on one machine then derives a configuration included in $\mathcal{K}$. We set the variables x_κ^u accordingly. Let $u \in [C]$. If u is a large class, it contains only large jobs. These jobs are split into pieces of size $\delta^3 T$ starting at multiplicities of $\delta^3 T$ defining the used modules for that class. We set the variables y_θ^u accordingly and the variables $z_{h,\bar{b}}^u$ to 0. Further, we can count the number of windows being used for one large processing time layer-wise from any class deriving the values for the variables $a_{p,\ell}^u$. If, on the other hand, u is a small class, then the whole class is scheduled on the same machine i. Let κ be the configuration corresponding to i, $h = \Lambda(\kappa)$, and $\bar{b} = k_i - \|\kappa\|_1$. We set $z_{h,\bar{b}}^u = 1$ and $z_{h',\bar{b}'}^u = 0$ for each $(h', \bar{b}') \neq (h, \bar{b})$. Further, we set all the variables y_θ^u to 0. It is easy to verify that this solution is feasible. □

Hence, if the n-fold ILP has no feasible solution, we can reject the makespan guess T.

SOLVING THE n-FOLD Again, we start with estimating the size of the generated sets and numbers:

- $|\mathcal{P}| = O(1/\delta^3)$ by the rounding,
- $|L| = O(1/\delta^3)$ due to the definition of layers,
- $|\Theta| = 2^{O(1/\delta^3)}$, as we have $1/\delta^3$ many layers and can chose for each of them if they are used in the module or not,
- $|\mathcal{K}| = (2^{O(1/\delta^3)})^{1/\delta^3} = 2^{O(1/\delta^6)}$, as we can chose at most $1/\delta^3$ modules from the set of all modules to not double use the same layer,
- $|\Lambda(\mathcal{K})| = O(1/\delta^3)$ due to the number of layers each of the same size,
- $c^* = O(1/\delta^3)$, as at most $1/\delta^3$ job pieces fit on a machine, and
- $|\mathcal{B}| = O(\nu/\delta^3)$, as 0 up to $1/\delta^3$ class slots can be occupied by large jobs (job pieces) leaving $1/\delta^3 + 1$ different values for the remaining class slots for each machine and we have ν machine types with different numbers of initial free class slots

This yields the following dimensions of the n-fold ILP:

- $n = C$,
- $r = |\Theta| + |\Lambda(\mathcal{K})| \cdot |\mathcal{B}| = 2^{O(\nu/\delta^3)}$,
- $s = |\mathcal{P}| + |L| = O(1/\delta^3)$, and
- $t = |\mathcal{K}| + |L| + |\Theta| + |\mathcal{P}| + |\Lambda(\mathcal{K})| \cdot |\mathcal{B}| = \nu 2^{O(1/\delta^6)}$.

The largest absolute value of each number in the constraint matrix is upper bounded by $\bar{T}$. Due to the scaling and as $k_{\max} \leq C$ we have $\bar{T} = O(C/\delta^3)$. Applying Proposition 1 and using that $C \leq N$ gives us a running time to solve the n-fold ILP of:

$$2^{2^{O(\nu/\delta^3)}}(2^{O(\nu/\delta^3)}C/\delta^6)^{2^{O(\nu/\delta^3)}}(C\nu 2^{O(1/\delta^6)})^{1+o(1)} = N^{2^{O(\nu/\delta^3)}}$$

CONSTRUCTING THE SCHEDULE Again, we have to use the solution of the n-fold ILP to construct a schedule. First, we assign the configurations chosen by the variables x^u_κ onto the machines, i. e., create the corresponding windows of size $\delta^3 T$ at the layers of the belonging modules. Next, we reserve the windows for the corresponding classes according to the variables y^u_θ. Finally, we fill the job pieces belonging to large classes accordingly to the variables $a^u_{p,\ell}$ greedily by proceeding as follows: We go through the layers $\ell \in L$ in an arbitrary order. Fill $a^u_{p,\ell}$ many job pieces of $a^u_{p,\ell}$ different jobs with processing time p, class u and the most unassigned job pieces of size $\delta^3 T$ onto the machines which have windows reserved for that class. It is easy to verify that we have enough windows for placing the job in this manner due to the constraints of the n-fold ILP. Further, the next theorem proves that this approach assigns all large jobs without conflict, i. e., no job pieces belonging to the same large job are assigned to the same layer.

Theorem 58. *We can greedily assign jobs accordingly to the variables $a^u_{p,\ell}$ of class u with processing time p, respecting the variables y^u_θ, such that job pieces of the same class are not executed in parallel.*

Proof. Suppose the opposite. At some layer ℓ there are w.l.o.g. two windows but just one job j of class u, which has two job pieces left. This would imply that we placed the last job piece of another job of that class in some layer before while having two job pieces of j. This contradicts the procedure of the greedy algorithm. It remains to prove that this also cannot happen while filling the first layer. Indeed, this would imply that we only have one job of processing time p and class u and thus, Constraint (4) would only allow one placeholder in each layer. Altogether, this proves the theorem. □

Next, we assign the small jobs similar as before by using the round robin approach. For each $h \in \Lambda(\mathcal{K})$ and $\bar{b} \in \mathcal{B}$, we assign the jobs of small classes u with $z^u_{h,\bar{b}} = 1$ onto the machines with configurations where h space is used by large classes and $\bar{b}$ class slots are empty via round robin. Due to Constraint (2), all the jobs can be placed by this procedure. Furthermore, due to Lemma 41 and Constraint (3), this yields a schedule with makespan at most $\bar{T} + \delta T$. Lastly, we have to use the original running times and jobs, which can be done using a greedy approach.

The overall running time for placing the large classes is linear in the number of involved job pieces, that is $O(M/\delta^3) \leq O(N/\delta^3)$. When placing the small jobs, we consider each class at most once, i. e., it takes time $O(C)$. Further, when we insert the original jobs and job sizes, we have to consider each job and job piece in the schedule once. This yields an overall running time of $O(N/\delta^3)$.

TOTAL RUNNING TIME AND ERROR We have seen how we can solve the preemptive case of the VARIABLE CLASS CONSTRAINT SCHEDULING problem when we are given a guess T on the makespan. Following the idea of the algorithms above, we have to complete the algorithm by embedding it into a binary search. Again, $N \cdot p_{\max}$ states an upper bound. Further, using the fact that we are aiming for a $O(\delta)$ precision, the binary search is exhausted after at most $O(\log((N \cdot p_{\max}/\delta))$ steps. Using $C \leq N$, we get a total running time of:

$$\underbrace{O(\log(N \cdot p_{\max}/\delta))}_{\text{Binary Search}} \cdot [\underbrace{O(N)}_{\text{Preprocess}} + \underbrace{N^{2^{O(\nu/\delta^3)}}}_{n\text{-fold ILP}} + \underbrace{O(N/\delta^3)}_{\text{Schedule construction}}]$$
$$= N^{2^{O(\nu/\delta^3)}} \log(p_{\max}).$$

Setting $\delta = O(\epsilon)$, we get an overall error of ϵ as the error of each step is bounded by $O(\delta)$ as we argued above. This yields the complete algorithm and its analysis. Summarizing, we get:

Theorem 59. *A schedule σ for the preemptive case of the VARIABLE CLASS CONSTRAINED SCHEDULING problem with makespan $\mu(\sigma) \leq (1+\epsilon) \cdot \mu(\text{OPT}(I))$*

can be computed in time $N^{2^{O(\nu/\epsilon^3)}} \log(p_{\max})$*, where* $\textsc{opt}(I)$ *denotes a solution with optimal makespan for scheduling the instance* I *and* ν *is a parameter stating the number of machine types. This yields the desired PTAS for this problem.*

No dragon can resist the fascination of riddling talk and of wasting time trying to understand it.

— from *The Hobbit* by J. R. R. Tolkien

10

SOLVING PACKING PROBLEMS WITH FEW SMALL ITEMS

In the previous chapters, we studied different scheduling variants. In the following, we want to focus on another important area of combinatorial optimization, in particular, packing problems. Central among those is the Bin Packing problem, which has sparked numerous important algorithmic techniques. Recall that in the Bin Packing problem, the goal is to pack a set of N items with sizes in $(0,1]$ into as few unit-sized bins as possible.

Referring to its simplicity and vexing intractability, this problem has been labeled as "the problem that wouldn't go away" more than three decades ago [66] and is still the focus of groundbreaking research today. For example, it has been studied extensively from the viewpoint of approximation algorithms. The best known for any instance I of this problem is an additive $O(\log(\mathrm{OPT}(I)))$-approximation algorithm due to Hoberg and Rothvoß [70, 86].

Another more recent trend is to apply tools from parameterized complexity theory to problems from operations research [133], which, however, has received much less attention so far. For the Bin Packing problem, a natural parameter is the minimum number of bins. For this parameter, Jansen et al. [99] showed that this problem is $\mathsf{W}[1]$-hard, even for instances encoded in unary. Another natural parameter is the number d of distinct item sizes. For $d = 2$, a polynomial-time algorithm was discovered by McCormick et al. [132] in the 1990s. The complexity for all $d \geq 3$ was open for more than 15 years, until a breakthrough result of Goemans and Rothvoß [70] showed that Bin Packing can be solved in time $(\log(\Delta))^{2^{O(d)}}$, where Δ is the largest number in the input. A similar result was shown later by Jansen and Klein [94]. Neither the algorithm by Goemans and Rothvoß nor the algorithm by Jansen and Klein are fixed-parameter algorithms for parameter d, which would require the algorithm to run in time $f(d) \cdot \mathrm{poly}(|I|)^{O(1)}$ for some computable function f. However, the algorithm by Jansen and Klein is a fixed-parameter algorithm for the parameter $|V_I|$, i. e., the number of vertices of the integer hull of the underlying knapsack polytope.

In light of these daunting results, we propose another natural parameter for the Bin Packing problem. This parameter is motivated by the classical approach of parameters measuring the *distance from triviality*, a concept that was first proposed by Niedermeier [137, Sect. 5.4]. Roughly speaking, this approach measures the distance of the given instance from an instance which is solvable in polynomial time. This idea was already used for many different problems such as Clique, Set Cover, Power Dominating Set, or Longest

Common Subsequence [76]. Even one of the arguably most important graph parameters called *treewidth* is often interpreted as the distance of a given graph from a tree [76]. Interestingly, the number of special cases where the Bin Packing problem can be solved in polynomial time is rather limited. In this chapter, we propose as a novel parameter the distance from instances without *small items*. If no small item (with size at most $1/3$) exists, Bin Packing becomes solvable in polynomial time via a reduction to a matching problem, as each bin can contain at most two items. If the number of small items is unbounded, the problem becomes NP-hard.

We also investigate a related problem with respect to this parameter called Multiple Knapsack, which is a generalization of the Knapsack problem. The problem has been studied extensively (see the books by Gonzalez [73] and Kellerer et al. [110]). It shares the Bin Packing trait that the efficiency of exact algorithms is hindered by the existence of small objects.

In the aforementioned problems, the items have a one-dimensional size requirement. As this is too restrictive in many applications, so-called *vector versions* were proposed [5, 65]. In these versions, called Vector Packing, and Vector Multiple Knapsack, each object has a D-dimensional size requirement and a set of objects can be packed only if the size constraints are satisfied in *each* dimension $\ell = 1, \dots, D$. These problems are much harder than their 1-dimensional version, e. g., Vector Packing does not admit an asymptotic polynomial time approximation scheme even for $D = 2$ [165]. For the D-dimensional problems, we use the words *vectors* instead of *items* and *containers* instead of *bins*. In the following, we study the vector variants of these problems and thus, we first have to define the smallness of vectors accordingly.

In the one-dimensional version of Vector Packing, the definition of a small item is quite natural: Every item with size less or equal than $1/3$ is considered small, the remaining ones large. As a consequence, each bin can contain at most two large items. We would like to transfer this property from one dimension to the D-dimensional case, in particular, the requirement that at most two large vectors fit inside the same container. We call a subset $\mathcal{V}' \subseteq \mathcal{V}$ of vectors *3-incompatible* if no selection of three distinct vectors from $\mathcal{V}'$ may be placed in the same container, i.e., for each $u, v, w \in \mathcal{V}'$, there exists an $\ell \in \{1, \dots, D\}$ such that $u^\ell + v^\ell + w^\ell > T^\ell$, where T^ℓ is the *capacity constraint* of the container in dimension ℓ. Let $\mathcal{V}_L \subseteq \mathcal{V}$ be a *largest* 3-incompatible set. We call the vectors $v \in \mathcal{V}_L$ *large* and call the vectors from the set $\mathcal{V}_S = \mathcal{V} \setminus \mathcal{V}_L$ *small*. Note that each 3-incompatible set $\mathcal{V}'$ contains at most two vectors where all the entries have size of at most $1/3$. Hence, for Bin Packing the largest 3-incompatible set corresponds to the set of large items plus at most two additional items.

An important property of our definition is that the smallness of a vector is no longer an attribute of the vector itself, but needs to be treated with respect to all other vectors. Finding a set $\mathcal{V}_S \subseteq \mathcal{V}$ of small vectors of minimum cardinality might be non-trivial. We argue that this task is fixed-parameter tractable parameterized by $|\mathcal{V}_S|$. To find $\mathcal{V}_S$, we compute the largest 3-incompatible set

in $\mathcal{V}$. The complement $\mathcal{V}_S = \mathcal{V} \setminus \mathcal{V}_L$ of a largest 3-incompatible set can be found in time $f(|\mathcal{V}_S|) \cdot N^{O(1)}$ by a reduction to 3-Hitting Set. In this problem, a collection of sets $S_1, \ldots, S_{n_H} \subseteq U$ for some universe U with $|S_i| = 3$ is given, and a set $H \subseteq U$ of size at most $|\mathcal{V}_S|$ with $H \cap S_i \neq \emptyset$ for all $i \in \{1, \ldots, n_H\}$ is sought. In Section 10.1, we present a reduction from the problem of finding the sets $\mathcal{V}_L$ and $\mathcal{V}_S$ to an instance of the 3-Hitting Set problem, which we can solve using:

Proposition 60 ([20, 60, 138]). *3-Hitting Set can be solved in time $2.27^{k_H} \cdot n_H^{O(1)}$, where k_H is the size of the solution. A corresponding solution can be obtained within the same time.*

Overall, we settle the parameterized complexity of the vector versions of Bin Packing and Multiple Knapsack parameterized by the number k of small objects. Our main results are randomized fixed-parameter algorithms, which solve these problems in FPT with one-sided error. Note that Vector Multiple Knapsack is already NP-hard for $D = 1$ and $w_{\max} \leq N^{O(1)}$ [68, 129] where $w_{\max}$ denotes the largest profit of any object. Our approach is to reduce the vector versions of the packing problems to a new matching problem on edge-colored graphs, which we call Perfect Over-the-Rainbow Matching.

In the Perfect Over-the-Rainbow Matching problem, we are given a graph G. Each edge $e \in E(G)$ is assigned a set of colors $\chi(e) \subseteq \mathcal{C}$ and for each color, there is a non-negative weight $\omega(e, c)$. The objective is to find a perfect matching P (a subset of edges such that each node is covered by exactly one edge) of G and a function $\zeta : P \to \mathcal{C}$ such that

(i) $\zeta(e) \in \chi(e)$ for all $e \in P$ (we can only choose from the assigned colors),

(ii) $\bigcup_{e \in P} \zeta(e) = \mathcal{C}$ (every color is present in the matching), and

(iii) $\sum_{e \in P} \omega(e, \zeta(e))$ is minimized (the sum of the weights is minimized).

The parameter for the problem is $|\mathcal{C}|$, the number of different colors. We want to emphasize that the colored matching problem is natural in itself and we expect it to be useful for other applications.

To solve the Perfect Over-the-Rainbow Matching problem, we employ an approach that is based on the Conjoining Matching problem. The Conjoining Matching problem was proposed by Sorge et al. [157], who asked whether it is fixed-parameter tractable. The question was resolved independently by Gutin et al. [78] and by Marx and Pilipczuk [131], who both gave randomized fixed-parameter algorithms. Based on both results, we obtain the following:

Theorem 61. *There is a randomized algorithm (with bounded false negative rate in $n_R + \Upsilon_R$) that solves Perfect Over-The-Rainbow Matching in time $2^{|C|} \cdot n_R^{O(1)} \Upsilon_R^{O(1)}$, where n_R is the number of nodes and Υ_R is the desired profit.*

Whether there is a deterministic fixed-parameter algorithm for Conjoining Matching remains a challenging question as also pointed out by Marx

and Pilipczuk [131]. For some of the problems that can be solved by the randomized algebraic techniques of Mulmeley, Vazirani and Vazirani [135], no deterministic polynomial-time algorithms have been found, despite significant efforts. The question is whether the use of such matching algorithms is essential for Conjoining Matching, or can be avoided by a different approach.

We succeed in circumventing the randomness of our algorithm in the 1-dimensional case of Bin Packing. Namely, we develop a deterministic algorithm running in time $O((k!)^2 \cdot k \cdot 2^k \cdot N \log(N))$ for Bin Packing by proving strong structural properties of optimal solutions. Those structural insights may be of independent interest.

Before we finally present our algorithms, we give a short introduction to randomized algorithms. Further, we introduce the packing problems formally and define the auxiliary matching problem necessary in the following. As we introduce various problems in this chapter, the reader might return to the definitions. Thus. we highlight them by framing each one.

RANDOMIZED ALGORITHMS A *randomized algorithm* is an algorithm that explores some of its computational paths only with a certain probability. A randomized algorithm A for a decision problem L has *one-sided error* if it either correctly detects positive or negative instances with probability 1. It has a *bounded false negative rate* if $\Pr[\mathsf{A}(x) = \text{“no”} \mid x \in L] \leq 1/|x|^C$, that is, it declares a “yes”-instance as a “no”-instance with probability at most $1/|x|^C$ for some constant C. All randomized algorithms in this chapter have bounded false negative rate.

PACKING PROBLEMS In the Vector Packing problem, we aim to pack a set $\mathcal{V} = \{v_1, \ldots, v_N\} \subseteq \mathbb{Q}_{\geq 0}^D$ of vectors into the smallest possible number of containers, where all containers have a common capacity constraint $T \in \mathbb{Q}_{\geq 0}^D$. Let $v_j \in \mathcal{V}$ be a vector. We use v_j^ℓ to denote the ℓ^{th} component of v_j and T^ℓ to denote the ℓ^{th} constraint. A *packing* is a mapping $\rho\colon \mathcal{V} \to \mathbb{N}_{>0}$ from vectors to containers. It is *feasible* if all containers $i \in \mathbb{N}_{>0}$ meet the capacity constraint, that is, for each $\ell \in \{1, \ldots, D\}$ it holds that $\sum_{v_j \in \rho^{-1}(i)} v_j^\ell \leq T^\ell$. Using as few containers as possible means to minimize $\max\{\rho(v_j) \mid v_j \in \mathcal{V}\}$. We expect only few small items, so we consider this quantity as parameter for Vector Packing yielding:

Vector Packing	
Parameter:	Number k of small vectors
Input:	A set $\mathcal{V} = \{v_1, \ldots, v_N\} \subseteq \mathbb{Q}_{\geq 0}^D$ of vectors and a capacity constraint $T \in \mathbb{Q}_{\geq 0}^D$
Task:	Packing $\mathcal{V}$ into the smallest number of containers

The 1-dimensional case of the problem is the Bin Packing problem. There, vectors are called *items*, their single component *size* and the containers *bins*. In contrast to the multi-dimensional case, we are now given a *sequence* of

items, denoted as $\mathcal{I}$. This is due to the fact that in the multi-dimensional setting, we can simply model multiple occurrences of the same vector by introducing an additional dimension encoding the index of the vector. This is not possible in the one-dimensional case.

BIN PACKING	
Parameter:	Number k of small items
Input:	A sequence $\mathcal{I} = (s_1, \dots, s_N)$ of items such that $s_j \in \mathbb{Q}^1_{\geq 0}$ for each $s_j \in \mathcal{I}$ and a capacity constraint $T \in \mathbb{Q}^1_{\geq 0}$
Task:	Packing $\mathcal{I}$ into the smallest number of bins

Another related problem is the VECTOR MULTIPLE KNAPSACK problem. Here, a packing into M containers is sought. Due to the limited number of containers, not all vectors may fit into them. We have to choose which vectors we pack considering that each vector $v \in \mathcal{V}$ has an associated profit $w_v \in \mathbb{N}_{\geq 0}$. A packing of the vectors is a mapping $\rho\colon \mathcal{V} \to \{1, \dots, M\} \cup \{\perp\}$ such that $\sum_{v_j \in \rho^{-1}(i)} v_j^\ell \leq T^\ell$ holds for all $i \in \{1, \dots, M\}$ and $\ell \in \{1, \dots, D\}$, which means no container is overpacked. The objective is to find a packing with a maximum total profit of the packed items, that is, we want to maximize $\sum_{i=1}^{M} \sum_{v \in \rho^{-1}(i)} w_v$.

VECTOR MULTIPLE KNAPSACK	
Parameter:	Number k of small vectors
Input:	A set $\mathcal{V} = \{v_1, \dots v_N\} \subseteq \mathbb{Q}^D_{\geq 0}$ of vectors, their profits $w_{v_j} \in \mathbb{N}_{\geq 0}$, a capacity constraints $T \in \mathbb{Q}^D_{\geq 0}$ and the number of bins M
Task:	Packing $\mathcal{V}$ into M bins while maximizing the profit

RANDOMIZED ALGORITHM USING CONJOINED MATCHINGS We introduce two useful problems to tackle the questions mentioned above, namely PERFECT OVER-THE-RAINBOW MATCHING and CONJOINING MATCHING. The following section presents the reductions from VECTOR PACKING and VECTOR MULTIPLE KNAPSACK to CONJOINING MATCHING using PERFECT OVER-THE-RAINBOW MATCHING as an intermediate step. By the results of Gutin et al. [78] and Marx and Pilipczuk [131], we can solve CONJOINING MATCHING efficiently and thus, our packing problems as well. A *matching* in a graph G describes a set of edges $P \subseteq E(G)$ without common nodes, that is, $e_1 \cap e_2 = \emptyset$ for all distinct $e_1, e_2 \in P$. A matching is *perfect* if it covers all nodes.

In the PERFECT OVER-THE-RAINBOW MATCHING problem, we are given a graph G as well as a *color function* $\chi\colon E(G) \to 2^{\mathcal{C}} \setminus \{\emptyset\}$ which assigns a non-empty set of colors to each edge, and an integer Υ_R. For each edge e and each color $c \in \chi(e)$, there is a non-negative *weight* $\omega(e, c)$. The objective is to find a perfect matching P and a surjective function $\zeta\colon P \to \mathcal{C}$ with $\zeta(e) \in \chi(e)$ for each $e \in P$ such that $\sum_{e \in P} \omega(e, \zeta(e)) \leq \Upsilon_R$. The surjectivity guarantees that

each color has to appear at least once. We call such a pair (P, ζ) a perfect *over-the-rainbow matching* and the term $\sum_{e \in P} \omega(e, \zeta(e))$ denotes its weight. The problem has its origins in the closely related, eponymous *rainbow matching* problem, where each color appears exactly once [109, 123]. In contrast to our problem, a sought rainbow matching covers as many colors as possible, but not necessarily all, and the maximum size of a rainbow matching is bounded by the number of colors. In our variant, we have to cover all colors, and likely have to cover some colors more than once to get a perfect matching. Formally, the problem is defined as follows:

Perfect Over-the-Rainbow Matching	
Parameter:	The number of colors $\lvert\mathcal{C}\rvert$
Input:	A graph G, a set of colors $\mathcal{C} = \{1, \dots, \lvert\mathcal{C}\rvert\}$, a function $\chi \colon E \to 2^{\mathcal{C}} \setminus \{\emptyset\}$, edge weights $\omega : \{(e,c) \mid e \in E(G), c \in \chi(e)\} \to \mathbb{Q}_{\geq 0}$, and a number Υ_R
Task:	Find a perfect over-the-rainbow matching (P, ζ) in G of weight at most Υ_R

We sometimes omit the surjective function ζ, if it is clear from the context.

Related to this problem is Conjoining Matching. Here, we have a partition $V_1 \uplus \cdots \uplus V_t$ of the nodes of G and a pattern graph F with $V(F) = \{V_1, \dots, V_t\}$. Instead of covering all colors in a perfect matching, this problems asks to find a *conjoining* matching $P \subseteq E(G)$, which is a perfect matching such that for each $\{V_i, V_j\} \in E(F)$, there is an edge in P with one node in V_i and the other in V_j. Roughly speaking, each edge in P corresponds to some edges in G of which at least one has to be taken by P. Formally, we define:

Conjoining Matching	
Parameter:	The number of edges of F
Input:	A weighted graph $G = (V, E, \omega)$ with $\omega \colon E \to \mathbb{Q}_{\geq 0}$, a node partition $V_1 \uplus \cdots \uplus V_t$, a number Υ_C, and a graph F with $V(F) = \{V_1, \dots, V_t\}$
Task:	Find a perfect matching P in G of weight at most Υ_C such that for each edge $\{V_i, V_j\} \in E(F)$, there is an edge $\{u, v\} \in P$ with $u \in V_i$ and $v \in V_j$

Gutin et al. [78, Theorem 7] and Marx and Pilipczuk [131] gave randomized fixed-parameter algorithms for Conjoining Matching on loop-free graphs F. We show how a simple reduction also solves the problem on graphs with loops.

Lemma 62. *The Conjoining Matching problem can be solved by a randomized algorithm (with bounded false negative rate in $n_C + \Upsilon_C$) in time*

$2^{|E(F)|} n_C^{O(1)} \Upsilon_C^{O(1)}$, *even if* F *contains self-loops. Here,* n_C *denotes the number of nodes in* G.

Proof. If F does not contain self-loops, the claim is proven by Gutin et al. [78]. In the case that F does contain self-loops, this can be reduced to the loop-free version by a simple layering argument: First direct the edges of F arbitrarily (for instance by using the lexicographical order of the nodes) and then, define G' and F' as

$$\begin{aligned}
V(F') &= \{h', h'', h^* \mid h \in V(F)\},\\
E(F') &= \{\{h_i', h_j''\} \mid (h_i, h_j) \in E(F)\},\\
V(G') &= \{v', v'', v^* \mid v \in V(G)\},\\
E(G') &= \{\{v', v^*\}, \{v'', v^*\} \mid v \in V(G)\} \cup \{\{v', w''\} \mid \{v, w\} \in E(G)\}.
\end{aligned}$$

Observe that F' is loop-free and $|E(F)| = |E(F')|$. Further, note that in any perfect matching in G', for each $v \in V(G)$, either v' or v'' have to be matched with v^*. The other node together with its matching partner corresponds to an edge in a corresponding perfect matching in G, as it is only connected to v^* or $\{w', w'' \mid \{v, w\} \in E(G)\}$. Finally, to preserve weights, set $\omega'(\{v', v^*\}) = \omega'(\{v'', v^*\}) = 0$ and $\omega'(\{v', w''\}) = \omega(\{v, w\})$ for all $v, w \in V(G)$. □

SUMMARY OF RESULTS

- The Vector Packing problem can be solved by a randomized algorithm (with bounded false negative rate in N) in time $2^{O(k)} \cdot k! \cdot N^{O(1)}$.
- The Vector Multiple Knapsack problem can be solved by a randomized algorithm (with bounded false negative rate in $N + w_{\max}$) in time $2^{O(k)} \cdot k! \cdot N^{O(1)} \cdot (w_{\max})^{O(1)}$ where $w_{\max}$ is the largest profit of any vector.
- There is a randomized algorithm (with bounded false negative rate in $n_R + \Upsilon_R$) that solves the Perfect Over-The-Rainbow Matching problem in time $2^{|\mathcal{C}|} \cdot n_R^{O(1)} \cdot \Upsilon_R^{O(1)}$.
- The Bin Packing problem can be solved deterministically in time $O((k!)^2 \cdot k \cdot 2^k \cdot N \log(N))$ using structural properties of optimal solutions.

FURTHER RELATED WORK The class of small items, their relation to matching problems, and special instances without small items have been extensively studied in the literature: Shor [154, 155] studies the relation between *online bin packing*, where the items are uniformly randomly chosen from $(0, 1]$, and the Matching problem on planar graphs. Those problems are closely related, as almost all bins in an optimal solution contain at most two items. Csirik et al. [38] study the Generalized First-Fit-Decreasing heuristic for Vector Packing and show that their strategy is optimal for instances that contain at most two small items. Kenyon [113] studies the expected

performance ratio of the *Best-Fit* algorithm for Bin Packing on a worst-case instance where the items arrive in random order. To prove an upper bound on the performance ratio, she classifies items into small items (size of at most $1/3$), medium items (size of at least $1/3$ and of at most $2/3$), and large items (size of at least $2/3$) [113]. Kuipers [121] studies so-called *bin packing games* where the goal is to share a certain profit in a fair way between the players controlling the bins and players controlling the items. He only studies instances without small items and shows that every such instances has a non-empty ε-core (a way of spreading the profits relatively fair) for $\varepsilon \geq 1/7$. Babel et al. [12] present an algorithm with competitive ratio $1 + 1/\sqrt{5}$ for online bin packing without small items. In another online version of the problem, the items and a conflict graph on them are given offline and then, in an online fashion, variable-sized bins arrive. The case that the conflict graph is the union of two cliques corresponds to instances with no small items and was studied by Epstein et al. [52]. Another version of Bin Packing forbids the packing of more than k different items into a single bin. The special case $k = 2$ corresponds to instances without small items and can be solved in time $N^{O(1/\varepsilon^2)}$ for bins of size $1 + \varepsilon$ [53]. Further, Bansal et al. [15] study approximation algorithms for Vector Packing. To obtain their algorithms, they present a structural lemma that states that any solution with M bins can be turned into a solution with $(D+1)M/2$ bins such that each bin either contains at most two items or has empty space left in all but one dimensions. This result is then used to reduce the problem to a Multi-Objective Budgeted Matching problem.

From an approximation point of view, the problems considered in this work have been studied extensively, both for the 1-dimensional variant as well as for the vector versions. We refer to the survey of Christensen et al. [32] for an overview. Regarding parameterized algorithms for problems from operations research, the resulting body of literature is too large for a detailed description and we refer to the survey of Mnich and van Bevern [133]. Some of the 1-dimensional variants of the problems considered in this work have been studied from a parameterized perspective [94, 99, 105]. In contrast, to the best of our knowledge, there are no such results for the vector versions of these problems.

STRUCTURE OF THIS CHAPTER In Section 10.1, we show the parameterized reductions from the packing problems to the Perfect Over-the-Rainbow Matching problem. Afterwards, we present the parameter preserving transformation of the Perfect Over-the-Rainbow Matching problem to Conjoining Matching and the resulting parameterized algorithms. As the algorithm for Perfect Over-the-Rainbow Matching is randomized, so are the algorithms for the packing problems. In Section 10.2, we give a deterministic parameterized algorithm for the classical 1-dimensional version of the Bin Packing problem.

10.1 RANDOMIZED ALGORITHMS USING CONJOINED MATCHINGS

In this section, we prove that VECTOR PACKING can be solved in time $2^{O(k)} \cdot k! \cdot N^{O(1)}$ and the MULTIPLE KNAPSACK problem can be solved in time $2^{O(k)} \cdot k! \cdot N^{O(1)} \cdot (w_{\max})^{O(1)}$ respectively. The first step to solve these packing problems is to interpret them as PERFECT OVER-THE-RAINBOW MATCHING problems. Each problem admits a similar procedure: Guess the packing of the small vectors; guess the number of large vectors for each container; use these guesses to pack the large vectors by formulating the problem as a matching problem in a graph. The idea is that the nodes of this graph represent the large vectors. An edge represents that both endpoints fit into the same container. Introducing a color and a weight function for the edges, we manage to handle the containers already filled with some small vectors and the overall profit of the packing. Note that the guessing also serves as a transformation from the minimization and maximization problems to decision problems, as each guess also corresponds to some fixed number of containers and, if applicable, to the profit. So, we ask if there is a solution with these numbers and thus, we can solve this question via such a reduction. In the second phase, we reduce the PERFECT OVER-THE-RAINBOW MATCHING problem to the CONJOINING MATCHING problem. Applying Lemma 62 results in a parameterized algorithm for finding perfect over-the-rainbow matchings, finalizing the algorithms for the allocation problems.

Reduction to the Perfect Over-The-Rainbow Matching problem

Before we can proceed as mentioned above, we first need to identify the sets $\mathcal{V}_L$ and $\mathcal{V}_S$ of large and small vectors explicitly. This can be done via a reduction to the 3-HITTING SET problem as follows: The set of elements is given by the set of vectors $\mathcal{V}$ and we compute all sets $S_i \subseteq \mathcal{V}$ of triplets that fit together in a single container, i. e., $|S_i| = 3$ and $\sum_{v \in S_i} v \leq T$. Consider a hitting set H for this instance. Then, the set $\mathcal{V} \setminus H$ is large, i. e., it is 3-incompatible. To see this, consider any three distinct vectors $u, v, w \in \mathcal{V} \setminus H$. If we had $u + v + w \leq T$, then the set $\{u, v, w\}$ would be part of the computed selection of subsets. Yet, $\{u, v, w\} \cap H = \emptyset$, which is a contradiction. We set the given number of small vectors k as k_H and use Proposition 60 to obtain a hitting set $H \subseteq \mathcal{V}$ of size at most $k = k_H$. We set $\mathcal{V}_L = \mathcal{V} \setminus H$ and $\mathcal{V}_S = H$. As there are $O(N^3)$ sets of triplets, this yields a run time of $2.27^k \cdot N^{O(1)}$ (see Proposition 60).

THE CASE OF PACKING VECTORS Recall that in the VECTOR PACKING problem, we are given N vectors of dimension D and a set of containers, each with the same size limitation $T \in \mathbb{Q}^D_{\geq 0}$. Furthermore, we assume that the sets $\mathcal{V}_S$ and $\mathcal{V}_L$ are given explicitly by using the computation explained above. Any solution needs at most $|\mathcal{V}|$ and at least $\lceil |\mathcal{V}_L|/2 \rceil$ containers. Furthermore, if there is a solution with $M \leq |\mathcal{V}|$ containers, there is also a solution with M' containers for any $M' \in \{M+1, \ldots, |\mathcal{V}|\}$. Thus, a binary search for the

optimal number of containers between the given bounds is possible. Let M be the current guess of the number of containers. Now, we have to decide whether there exists a solution using exactly M containers.

We guess the packing of the small vectors, that is, we try all possible partitions into at most $\min\{M,k\}$ subsets. It is not hard to see that the number of such partitions is upper bounded by the k^{th} Bell number: The first vector is packed by itself, the second can either be packed with the first one or also by itself, and so on. If any of the corresponding containers is already overpacked, we discard the guess. In the following, we call the used containers *partially filled* as some area is already occupied by small vectors. For these partially filled containers, we guess which of them are finalized, i. e., which of them do not contain an additional large vector in the optimal solution, and discard them for the following steps. There are at most 2^k such guesses. We denote the number of discarded containers as M_0. For each of the remaining partially filled containers, we introduce a new color. Furthermore, we introduce a color $\top$ representing the empty containers if existent. Hence, the resulting set of colors $\mathcal{C}$ has a cardinality of at most $k+1$. For each $c \in \mathcal{C}$, we denote by $s(c) \in \mathbb{Q}^D_{\geq 0}$ the residual size in the corresponding container.

We place the large vectors $\mathcal{V}_L$ inside the $M - M_0$ residual containers by reducing it to a Perfect Over-the-Rainbow Matching problem. Note that if the current guess is correct, each of the $M - M_0$ containers receives at least one and at most two large vectors. Hence, we may assume $|\mathcal{V}_L|/2 \leq (M - M_0) \leq |\mathcal{V}_L|$ (and reject the current guess otherwise). Furthermore, the number of containers receiving one or two large items respectively is already determined by M and M_0. We denote these numbers by M_1 and M_2. Remark that $M_2 = |\mathcal{V}_L| - (M - M_0) \geq 0$ and $M_1 := (M - M_0) - M_2 = 2(M - M_0) - |\mathcal{V}_L| \geq 0$.

We now construct a graph $G = (V, E)$ to find a feasible packing. Every large vector $v \in \mathcal{V}_L$ is represented by two nodes v and v' in V. Let $\mathcal{V}'_L = \{v' \mid v \in \mathcal{V}_L\}$. Next, we define a set $\mathcal{B}$ of $2 \cdot M_2$ new nodes called *blocker nodes*, which ensures that all vectors are placed inside exactly $(M - M_0)$ containers. We define $V := \mathcal{V}_L \cup \mathcal{V}'_L \cup \mathcal{B}$. In this graph, an edge between the nodes in $\mathcal{V}_L \cup \mathcal{V}'_L$ represents a possible packing of the large vectors inside one container. Hence, we add an edge $e = \{v, w\}$ between two original vectors $v, w \in \mathcal{V}_L$ and assign this edge some color $c \in \mathcal{C}$ if these vectors fit together inside the corresponding container. Furthermore, we add an edge between a vector $v \in \mathcal{V}_L$ and its copy $v' \in \mathcal{V}'_L$ and assign it the color $c \in \mathcal{C}$ if the vector alone fits inside the corresponding container. More formally, we introduce the set of edges $E_c := \{\{u, v\} \mid u, v \in \mathcal{V}_L, u + v \leq s(c)\} \cup \{\{v, v'\} \mid v \in \mathcal{V}_L, v \leq s(c)\}$ for each color $c \in \mathcal{C}$. Additionally, we insert an edge between each of the copied nodes $\mathcal{V}'_L$ and the blocker nodes $\mathcal{B}$. More formally, we define $E_\perp := \{\{v', b\} \mid v' \in \mathcal{V}'_L, b \in \mathcal{B}\}$. Together, we get $E := E_\perp \cup \bigcup_{c \in \mathcal{C}} E_c$. Finally, we define the color function χ with $\chi \colon E \to 2^{\mathcal{C} \cup \{\perp\}}$, such that each edge in E_c gets color c for each $c \in \mathcal{C}' := \mathcal{C} \cup \{\perp\}$. More formally, we define $\chi(e) := \{c \in \mathcal{C} \cup \{\perp\} \mid e \in E_c\}$. See Figure 10.1 for an example of the construction. Note that the weights on the edges are irrelevant in this case and can be set to

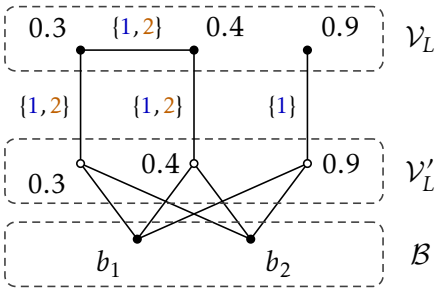

Figure 10.1: Construction of the graph G for a Bin Packing instance with sets $\mathcal{V}_S = \{0.1, 0.15, 0.2\}$ and $\mathcal{V}_L = \{0.3, 0.4, 0.9\}$. The guessed number of bins is $M = 3$. All small items are packed separately and the bin containing 0.15 is finalized ($M_0 = 1$). Thus, there is a bin containing 0.1 associated with color 1 (the first value in the braces) and a bin containing 0.2 associated with color 2 (the second value in the braces). The color $\perp$ used between all nodes of $\mathcal{V}'_L$ and all nodes of $\mathcal{B}$ is omitted.

1, i. e., $\omega(e, c) = 1$. To finalize the reduction, we have to define the size Υ_R of the matching we are looking for. We aim to find a perfect matching and hence, are searching for a matching of size $\Upsilon_R := |\mathcal{V}_L| + M_2$. Note that if $M_2 = 0$, no blocker nodes are introduced. We thus remove the color $\perp$ from the set of colors.

Lemma 63. *There is a packing of the large vectors $\mathcal{V}_L$ into $(M - M_0)$ containers such that each container holds at least one large vector if and only if the above described instance for Perfect Over-The-Rainbow-Matching is a "yes"-instance.*

Proof. Assume there is a packing of the vectors $\mathcal{V}_L$ into $(M - M_0)$ containers such that each container holds at least one large vector. In this case, we can construct a perfect over-the-rainbow matching M as follows. For each pair of vectors $v, w \in \mathcal{V}_L$ that is assigned to the same container, we choose the corresponding edge $\{v, w\}$ for the matching and assign it the corresponding color $c \in \mathcal{C}$. For each vector $v \in \mathcal{V}_L$ that is the only large vector in its container, we choose the edge $\{v, v'\}$ for the matching and assign it the corresponding color $c \in \mathcal{C}$. At this point, all the vectors in $\mathcal{V}_L$ are covered by exactly one matching edge since each of them is contained in exactly one container.

Note that in the given packing there have to be exactly $M_1 = 2(M - M_0) - |\mathcal{V}_L|$ containers with exactly one large vector and $M_2 = |\mathcal{V}_L| - (M - M_0)$ containers with exactly two large vectors. As a consequence, there are exactly $2 \cdot M_2$ nodes in $\mathcal{V}'_L$ that are not yet covered by a matching edge since their originals are covered by edges between each other. For each of these nodes, we choose an individual node from the set $\mathcal{B}$ and define the edge between these nodes as a matching edge and assign it the color $\perp$. Since there are exactly $2 \cdot M_2$ blocker nodes, we cover all nodes in V with matching edges and hence, we have constructed a perfect matching. Each color $c \in \mathcal{C}$ is represented by one partially filled container and hence, each has to appear in the matching. Moreover, if the color $\top$ was introduced, that is, there were less than $M - M_0$ containers partially covered by small vectors, then

there was a container exclusively containing large vectors and hence, $\top$ was used in the matching as well. Therefore, we indeed constructed a perfect over-the-rainbow matching.

Conversely, assume that we are given a perfect over-the-rainbow matching P. Consequently, each vector in $\mathcal{V}_L$ is covered by exactly one matching edge. As P contains at most $|\mathcal{V}_L| + M_2$ edges, and $2 \cdot M_2$ edges are needed to cover the nodes in $\mathcal{B}$, there are exactly $|\mathcal{V}_L| - M_2 = (M - M_0)$ matching edges containing the nodes from $\mathcal{V}_L$. As each color is present in P, we can represent each container by such a matching edge and place the corresponding vector or vectors inside the corresponding containers. If a color $c \in \mathcal{C}$ appears more than once, we use an empty container for the corresponding large vectors. □

To decide if there is a packing into at most M containers, we find a partition of the k small vectors with $O(k!)$ guesses, and the to-be-discarded containers with $O(2^k)$ guesses. Constructing the graph G needs $O(N^2 k)$ operations. By Theorem 61, a perfect over-the-rainbow matching over $k + O(1)$ colors with weight $\Upsilon_R \in O(N)$ can be computed in time $2^k \cdot N^{O(1)}$. To find the correct M, we call the above algorithm in binary search fashion $O(\log(N))$ times, as we need at most N containers. This results in a run time of $2^{O(k)} \cdot k! \cdot N^{O(1)}$.

THE CASE OF PACKING VECTORS WITH PROFITS Recall that in the VECTOR MULTIPLE KNAPSACK problem, we are given a set $\mathcal{V}$ of N vectors with dimension D, profits w_v for each $v \in \mathcal{V}$, and M containers each with capacity constraint $T \in \mathbb{Q}^D_{\geq 0}$. Furthermore, we are given a partition of the vectors $\mathcal{V}$ into a set of small vectors $\mathcal{V}_S$ and a set of large vectors $\mathcal{V}_L$.

Again, we guess the distribution of the small vectors. However, since it might not be optimal to place all the small vectors, we first have to guess which subset of them is chosen in the optimal solution. There are at most $2^k \cdot O(k!)$ possibilities for both guesses. After this step, we have at most k containers which are partially filled with small vectors.

In the next step, we guess for each partially filled containers whether they contain an additional large vector and discard the containers that do not. There are at most 2^k possible choices for this. Let M_0 be the number of such discarded containers. This step leaves $M - M_0$ containers for the large vectors in $\mathcal{V}_L$. Again, we define a color for each remaining partially filled container and one color $\top$ for the empty containers resulting in a set $\mathcal{C}$ of at most $k + 1$ colors.

Similar as for VECTOR PACKING, we construct a graph $G = (V, E)$ to find the packing which maximizes the profit. We introduce one node for each vector in $v \in \mathcal{V}_L$ and a node for its copy $v' \in \mathcal{V}'_L$. As before, we further introduce a set $\mathcal{B}$ of $2 \cdot |\mathcal{V}_L| - 2 \cdot (M - M_0)$ blocker nodes to ensure that we use exactly $M - M_0$ containers. We define a profit of zero for the copy nodes and the blocker nodes, while the nodes for the original vectors $v \in \mathcal{V}_L$ have profit w_v.

We add an edge between two nodes $v, w \in \mathcal{V}_L$ and assign it the color $c \in \mathcal{C}$ if the vectors together fit inside the corresponding container assigned

with color c. Furthermore, we add an edge between a node $v \in \mathcal{V}_L$ and its copy $v' \in \mathcal{V}'_L$ and assign it the color $c \in \mathcal{C}$ if it fits alone inside the corresponding container. More formally, we define for each color $c \in \mathcal{C}$ the set $E_c := \{\{u,v\} \mid u,v \in \mathcal{V}_L, u+v \leq s(c)\} \cup \{\{v,v'\} \mid v \in \mathcal{V}_L, v \leq s(c)\}$. Finally, we connect each node from the set $\mathcal{B}$ with each node from the set $\mathcal{V}_L \cup \mathcal{V}'_L$, i.e, we define $E_\perp := \{\{v,b\} \mid v \in \mathcal{V}_L \cup \mathcal{V}'_L, b \in \mathcal{B}\}$. In total, we set $E := E_\perp \cup \bigcup_{c \in \mathcal{C}} E_c$.

Finally, we define the color function χ and the profit function ω. For this purpose, we denote by $w_{\max}$ the maximal profit among the large vectors and define $\chi \colon E \to 2^{\mathcal{C} \cup \{\perp\}}$ with $e \mapsto \{c \mid c \in \mathcal{C} \cup \{\perp\}, e \in E_c\}$ as well as

$$\omega(\{v,u\},c) := \begin{cases} 2w_{\max} - (w_v + w_u), & v,u \in \mathcal{V}_L \cup \mathcal{V}'_L, \\ 0, & \text{otherwise,} \end{cases}$$

for all $e = \{v,u\} \in E$ and $c \in \mathcal{C} \cup \{\perp\}$ with $c \in \chi(e)$. Obviously, all the weights are non-negative.

Lemma 64. *There is a packing of the large vectors inside the corresponding $M - M_0$ containers with profit at least p if and only if there is a perfect over-the-rainbow matching P in G with weight at most $(M - M_0)w_{\max} - p$.*

Proof. Assume we are given a packing of large vectors inside the $M - M_0$ containers with profit at least p. For each packing of large vectors inside one container, we choose the edge between the corresponding pair of vectors (or between the vector and its copy in the case that the container has only one vector) for the matching and assign it the corresponding color. Now, there are exactly $2 \cdot (M - M_0)$ nodes in $\mathcal{V}_L \cup \mathcal{V}'_L$ covered by the matching. The remaining $|\mathcal{V}_L \cup \mathcal{V}'_L| - 2 \cdot (M - M_0)$ nodes in $\mathcal{V}_L \cup \mathcal{V}'_L$ are paired with one arbitrary node in $\mathcal{B}$. Since $\mathcal{B}$ contains exactly $2|\mathcal{V}_L| - 2 \cdot (M - M_0)$ nodes, each node can be paired.

The obtained matching P is a perfect matching since each node is covered. Furthermore, each color in $\mathcal{C} \cup \{\perp\}$ is used. Let $\mathcal{V}_{L,S} \subseteq \mathcal{V}_L$ be the set of large vectors packed in the given solution. By definition of the solution, it holds that the sum of profits for the vectors in $\mathcal{V}_{L,S}$ is greater than p. Note that the weight of an edge between two nodes $v,u \in \mathcal{V}_L$ is given by $2w_{\max} - (w_v + w_u)$, while edges between a node $v \in \mathcal{V}_L$ and its copy v' have the weight $2w_{\max} - w_v$. All the edges to the blocker nodes $\mathcal{B}$ have weight 0. Hence, the weight of the matching is given by $2(M - M_0)w_{\max} - \sum_{v \in \mathcal{V}_S} w_v \leq 2(M - M_0)w_{\max} - p$, which proves the first implication.

To prove the other direction, assume that we are given a perfect over-the-rainbow matching with weight at most $2(M - M_0)w_{\max} - p$. Each of the $2|\mathcal{V}_L| - 2 \cdot (M - M_0)$ blocker nodes in $\mathcal{B}$ is matched to exactly one node in $|\mathcal{V}_L \cup \mathcal{V}'_L|$. As a result, there are exactly $2 \cdot (M - M_0)$ nodes in $|\mathcal{V}_L \cup \mathcal{V}'_L|$ that are paired by the matching. We place the corresponding vectors inside the corresponding containers with respect to the color of the matching edge. If a color $c \in \mathcal{C}$ appears more than once, we use an empty container.

This packing is valid since each color appears at least once and hence, we can fill each container. Again, $\mathcal{V}_{L,S} \subseteq \mathcal{V}_L$ is the set of large vectors that

are matched with a node from the set $\mathcal{V}_L \cup \mathcal{V}'_L$. Then, by definition of the weight function, the matching has a size of $2(M - M_0)w_{\max} - \sum_{v \in \mathcal{V}_{L,S}} w_v \leq 2(M - M_0)w_{\max} - p$. As a consequence, the profit of the packing is given by $\sum_{v \in \mathcal{V}_{L,S}} w_v \geq p$. □

We can summarize the steps of the algorithm as follows. For each choice of small items and each possibility to distribute these items, the algorithm considers each choice of partially filled containers that do not contain an additional large item. For each of these choices, the algorithm constructs the graph described above. Then, it performs a binary search for a perfect over-the-rainbow matching with the smallest possible weight in the bounds $[0, 2(M - M_0)w_{\max}]$. Finally, it returns the packing with the largest total profit found among all possibilities.

By Theorem 61, we need at most $2^{|\mathcal{C} \cup \{\perp\}|} \cdot N^{O(1)} \cdot (w_{\max})^{O(1)} = 2^{k+2} \cdot N^{O(1)} \cdot (w_{\max})^{O(1)}$ operations to solve the constructed Perfect Over-the-Rainbow Matching problem. Finding the correct choice and partition of the k small vectors can be done in $O(2^k \cdot k!)$ guesses. Finding the containers without a large vector can be done in $O(2^k)$ guesses. Finally, the construction of the graph G needs at most $O(N^2 k)$ operations. The binary search procedure over the profits can be done in at most $O(\log((M - M_0)w_{\max})) = O(\log(N) + \log(w_{\max}))$ operations since the number of containers is bounded by $N^{O(1)}$. Hence, the run time is $2^{O(k)} \cdot k! \cdot N^{O(1)} \cdot (w_{\max})^{O(1)}$.

Reduction to the Conjoining Matching problem

Earlier, we have reduced both packing problems to Perfect Over-the-Rainbow Matching problems. Of course, all this effort would be in vain without the means to find such matchings efficiently. Next, we present a reduction to the task of finding a conjoining matching, which results in a parameterized algorithm for finding perfect over-the-rainbow matchings by applying Lemma 62. Overall, this proves Claim 61, which is repeated below for convenience:

Claim of 61. *There is a randomized algorithm (with bounded false negative rate in $n_R + \Upsilon_R$) that solves Perfect Over-The-Rainbow Matching in time $2^{|C|} \cdot n_R^{O(1)} \Upsilon_R^{O(1)}$, where n_R is the number of nodes and Υ_R is the desired profit.*

We aim to construct graphs F' and G' such that G has a perfect over-the-rainbow matching if and only if G' has a perfect conjoining matching of the same weight with respect to F'. Recall that in an over-the-rainbow matching, we request an edge of every color to be part of the matching, while in a conjoining matching, we request edges between certain sets of nodes to be part of the matching. For the reduction, we transform G into $G_1, \ldots, G_{|\mathcal{C}|}$ where each G_c is a copy of G containing only edges of color c. We set $V(G_c) = \{v_c \mid v \in V(G)\}$, i.e., v_c is the copy of $v \in V(G)$ in $V(G_c)$, and G_c contains only edges $e \in E(G)$ with $c \in \chi(e)$. We set G' to be the disjoint union of the G_c while setting $V(F') = \{V(G_c) \mid c \in \mathcal{C}\}$ and $E(F') =$

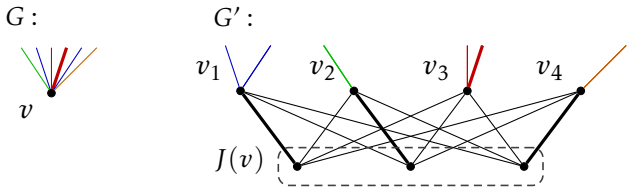

Figure 10.2: Reducing the problem of finding a perfect over-the-rainbow matching to the problem of finding a perfect conjoining matching. Left: single node of the colored input graph with a thick edge from a perfect matching. Right: $|\mathcal{C}| = 4$ copies of v in G'; the corresponding subgraphs G_c only contain edges of a single color. At the bottom, the added set $J(v)$ which is fully connected to all copies of v. The thick edges indicate how these nodes are paired in a perfect matching.

$\{\{h,h\} \mid h \in V(F')\}$. Now, a conjoined matching contains an edge of every color. However, the same edge of G could be used in multiple ways in the different copies G_c.

To address this issue, we introduce a gadget that enforces any perfect matching in G' to use at most one copy of every edge of G. In detail, for every node $v \in V(G)$, we add an independent set $J(v)$ of size $|\mathcal{C}| - 1$ to G'. Furthermore, we fully connect $J(v)$ to all copies of v in G', that is, we add the edges $\{v_c, x\}$ for all $c \in \mathcal{C}$ and $x \in J(v)$ to G'. This construction is illustrated in Figure 10.2. Observe that in any perfect matching of G', all elements of $J(v)$ have to be matched and thus, we "knock-out" $\left|J(v)\right| = |\mathcal{C}| - 1$ copies of v in G' leaving exactly one copy to be matched in one G_c. We add one more node to F' that represents the union of all the sets $J(v)$ and which has no connecting edge. To complete the description of the reduction, let us describe the weight function of G': For each $e \in E(G')$, we define

$$\omega'(e) := \begin{cases} \omega(e,c), & \text{if } e \in E(G_c) \text{ for some } c \in \mathcal{C} \\ 0, & \text{otherwise.} \end{cases}$$

Note that this definition implies that $\omega'(e) = 0$ for each e with $e \cap J(v) \neq \emptyset$ for some $v \in V(G)$.

Lemma 65. *Let $\left(G, \chi, \omega\right)$ be a colored and edge-weighted graph, and let G' and F' be defined as above. There is a perfect over-the-rainbow-matching P of weight Υ_R in G if and only if there is a perfect conjoining matching P' of weight Υ_C in G'.*

Proof. First, let us consider a perfect over-the-rainbow matching P in G. Let $\zeta\colon P \to \mathcal{C}$ be a surjective function with $\zeta(e) \in \chi(e)$ for all $e \in E(G)$. We show that there is a perfect conjoining matching in G'. Let $P' = \{\{v_{\zeta(e)}, w_{\zeta(e)}\} \mid e = \{v,w\} \in P\}$. Since P is a matching in G, P' is a matching in G' and since ζ is surjective, we have that P' contains at least one edge in every copy G_c of G in G' and thus, P' is actually a conjoining matching. Further, there is a bijection $f(e) = e_{\zeta(e)}$ between P and P' such that $\omega(e,\zeta(e)) = \omega'(f(e))$, so P and P' have the same weight.

By the definition of P', for every node $v \in V(G)$, there is exactly one color $c \in \mathcal{C}$ for that there is an edge in P' containing v_c. Therefore, the set $\{v_1, \ldots, v_{|\mathcal{C}|}\}$ contains exactly $|\mathcal{C}| - 1$ unmatched nodes for all $v \in V(G)$. We conclude that P' can be extended to a perfect conjoining matching P'' by paring these nodes with $J(v)$. Observe that P'' has the same weight as P', as the added edges have weight zero. Therefore, P'' has the same weight as P.

For the other direction, let us consider a perfect conjoining matching P' in G'. Observe that for all nodes $v \in V(G)$ the nodes in $J(v)$ have to be matched by P' and thus, for all nodes $v \in V(G)$, there is exactly one node $\alpha(v) \in \{v_1, \ldots, v_{|\mathcal{C}|}\}$ that is not matched with an element of $J(v)$. We define the set $P = \{\{v, w\} \mid v, w \in V(G) \text{ and } \{\alpha(v), \alpha(w)\} \in P'\}$ and claim that P is a perfect over-the-rainbow matching of the same weight as P'. First, observe that all $v \in V(G)$ are matched by P since P' is a perfect matching and thus, matches $\alpha(v)$ with, say, w_i. Observe that by the definition of α, we have $w_i \notin J(v)$ and by the construction of G', we have that w_i is a copy of some $w \in V(G)$ (it can, in particular, not be part of any other $J(u)$). Since w_i is paired with $\alpha(v)$, we conclude $\alpha(w) = w_i$ and thus, $\{\alpha(v), \alpha(w)\} \in P'$. Further, notice that every $v \in V(G)$ can be matched by at most one element of P, as P' is a perfect matching and thus, matches $\alpha(v)$ with exactly one other node. We conclude that P is a perfect matching of G. Finally, for all $\{v, w\} \in P$, observe that $\{\alpha(v), \alpha(w)\}$ has to lie in some copy G_c of G in G'. We define $\zeta(\{v, w\}) = c$ and $\omega(\{v, w\}, c) = \omega'(\{\alpha(v), \alpha(w)\})$. Observe that ζ is surjective since P' is conjoining and thus, witnesses that P is a perfect over-the-rainbow matching of G.

To conclude the proof, notice that P has the same weight as P', as for any edge of P' that has non-zero weight (that is, any edge that is not connected to some $J(v)$), we have added exactly one edge of the same weight to P. □

Proof of Theorem 61. Let $\left(G, \chi, \omega, \Upsilon_R\right)$ be an instance of Perfect Over-the-Rainbow Matching. We construct an instance $(G', F', \omega', \Upsilon_R)$ of Conjoining Matching in polynomial time, where the partition of $V(G')$ is defined as $V(G_1) \dot\cup V(G_2) \dot\cup \ldots \dot\cup V(G_{|\mathcal{C}|}) \dot\cup J$ with $J = \bigcup_{v \in V(G)} J(v)$ and $E(F')$ contains one self loop for each $V(G_c)$, $c \in \mathcal{C}$. It follows by Lemma 65 that $(G', F', \omega', \Upsilon_R)$ has a perfect conjoining matching of weight Υ_R if and only if $\left(G, \chi, \omega, \Upsilon_R\right)$ has a perfect over-the-rainbow matching of weight Υ_R. We apply Lemma 62 to find such a conjoining matching in time $2^{|E(F')|} n_R^{O(1)} \Upsilon_R^{O(1)}$. Observe that $|E(F')| = |\mathcal{C}|$ and thus, we can find the sought perfect over-the-rainbow matching in time $2^{|\mathcal{C}|} n_R^{O(1)} \Upsilon_R^{O(1)}$. □

10.2 DETERMINISTIC ALGORITHM FOR BIN PACKING

We now present a *fully-deterministic algorithm* for Bin Packing. The price we have to pay for circumventing the randomness is an increased running time, as we avoid the polynomial identity testing subroutine. On the bright side, this makes the algorithm more straightforward and simpler. We believe that extending this algorithm to Vector Packing is quite challenging. The

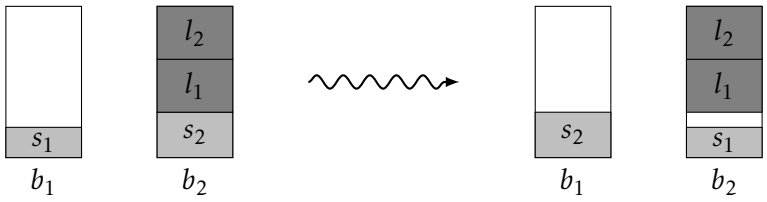

Figure 10.3: Proof of 66. The light gray rectangles denoted by s_1 and s_2 represent the load of small items on each of the two bins b_1 and b_2. The dark gray areas denoted with l_1 and l_2 represent large items in the bin b_2.

main obstacle is to identify the maximum item size in some sets, a task for which there does not seem to be a sensible equivalent notion for vectors.

ABOUT THE STRUCTURE OF OPTIMAL SOLUTIONS In the following, we prove the existence of an optimal solution that admits some useful properties regarding the placement of large items with respect to small ones. These properties are utilized in the algorithm later on.

Claim 66. *There is an optimal solution where the total size of small items on each bin containing only small items exceeds the total size of small items on each bin additionally containing large items.*

Proof. Suppose an optimal solution, in which the stated property is violated. Thus, there exist two bins b_1 and b_2 where the total size s_1 of small items on b_1 only admitting small items is smaller than the total size s_2 of small items on b_2 where also large items are placed, i. e., $s_1 < s_2$. We can now swap the sets of small items in b_1 and b_2. Since $s_1 < s_2$, the load of b_2 becomes smaller as it now contains small items with load $s_1 < s_2$. In turn, the total load on b_1 is now s_2. Since this entire set was placed on one bin before, b_1 is not overpacked. We can repeat this step until the property is satisfied for all bins. The proof is illustrated in Figure 10.3. □

Claim 67. *Given an optimal solution and an arbitrary order of the bins that each contain small items and exactly one large item. We can repack these large items correctly using a largest fitting approach with respect to the order of the bins. In detail, we greedily place the largest fitting item into the current bin.*

Proof. Consider the bins containing small items and exactly one large item in some given order. If the current bin b contains the largest fitting item regarding all items being packed on the later bins regarding the order, we consider the next bin. Otherwise, we swap the item i_b inside this bin with the largest item $i_{\max}$ that fits inside this bin and was not already placed inside a bin that was considered before. Note that the size of i_b has to be at most the size of $i_{\max}$ since $i_{\max}$ is the largest item that fits inside b, i. e., $i_b \leq i_{\max}$. As a consequence, no bin is overpacked after this swap since the total size of the items inside the other bin decreases or stays the same. □

Claim 68. *Consider an optimal solution where each partially filled bin contains exactly two items. Partially filled denotes again those bins where small items are placed in, occupying parts of the capacity. Let i_s be the smallest large item and i_ℓ be the largest one and let them fit together inside some partially filled bin. Then, there exists an optimal solution in which i_s is positioned inside a partially filled bin together with the largest large item that does fit additionally.*

Proof. Consider an optimal solution and the bin containing the smallest large item i_s. If i_s is positioned inside a partially filled bin, we can swap the additional large item with the largest item that fits together with i_s into this bin. This swap replaces an item inside one other bin with a smaller item, so the total size of items inside this bin decreases and hence, no bin is overpacked.

If i_s is not positioned inside a partially filled bin, then it fits together with the other large item it is currently paired with into a partially filled bin since we assumed that i_s even fits together with the largest item i_l into a partially filled bin. We swap this pair with the two large items of one (arbitrary) fitting partially filled bin. After this swap, no bin is overpacked since the other two large items fit in a partially filled bin. Hence, they fit inside an empty bin as well. Finally, we swap the item that is currently paired with the small item with the largest item that fits inside this bin together with i_s. As seen above, after this swap, there is no bin that is overpacked. □

Claim 69. *Consider an instance I where the largest large item i_ℓ does not fit together with the smallest large item i_s inside any partially filled bin and there is an optimal solution, where all partially filled bins contain exactly two large items. Then, there is an optimal solution which places i_ℓ together with the largest fitting large item inside one bin, or i_ℓ is placed alone inside a bin if there is no such large item.*

Proof. Consider an optimal solution, where each partially filled bin contains exactly two large items. Further, i_ℓ and i_s do not fit together inside a partially filled bin. Consider the bin b_1 containing the item i_ℓ. Clearly, i_ℓ is not contained inside a partially filled bin since it does not fit together with the smallest large item inside such a bin. Hence, it cannot fit together with any other large item inside a partially filled bin. Consider the largest item i that does fit together with i_ℓ inside one (empty) bin and let b_2 be the bin containing the item i. We can swap the item i_+ (if existent) that is currently placed together with i_ℓ with the item i. As item i_+ has at most the size of item i, bin b_2 is not overpacked by this step. On the other hand, as i_ℓ and i fit together inside a bin and there is no small item inside b_1, this bin is also not overpacked. If there is no large item that fits together with i_1 inside one bin in the case that b_1 does not contain any small items, i_1 is contained alone inside its bin. □

THE ALGORITHM In the first step of the algorithm, we sort the items according to their sizes in time $O(N \log(N))$. Next, we guess the distribution of the small items. Since there are at most k small items, there are at

most $O(k!)$ possible guesses. Again, we call the bins containing small items partially filled bins. There are at most k of these bins.

Then, we guess a bin b_1 that does not contain any additional large items All the partially filled bins containing small items with a larger total size than b_1 do not contain any large item as well, see Claim 66. Thus, we can discard them from the following considerations. There are at most k possibilities for the guess of b_1.

Now, we guess which of the remaining partially filled bins only contain one large item. There are at most $O(2^k)$ possibilities. We consider all partially filled bins for which we guessed that they only contain one large item in any order and pair them with the largest fitting item. By Claim 67, we know that an optimal packing with this structure exists. Afterwards, we discard these bins from the following considerations.

It remains to pack the residual large items. Each remaining partially filled bin contains exactly two large items in the optimal solution, otherwise the guess was wrong. To place the correct large item, we proceed as follows: Iterate through the large items in non-ascending order regarding their sizes. Let i_ℓ be the currently considered item. Further, let i_s be the smallest large item from the set of large items that still need to be placed. Depending on the relation between i_ℓ and i_s, we place at least one of these two items inside a bin. For the first case, it holds that i_ℓ does not fit together with i_s inside a partially filled bin. Then, we place i_ℓ together with the largest fitting item i from the set of large items that are not already placed inside one empty bin or place it alone inside an empty bin if such an item does not exist. The item i can be found, or its non-existence be proven in time $O(\log(N))$. For the second case, it holds that i_ℓ together with i_s does fit inside one partially filled bin. Then, we guess which partially filled bin contains i_s and place it inside this bin together with the largest unplaced item that fits inside this bin. The largest fitting item can be found in time $O(\log(N))$ and there are at most $O(k!)$ possible guesses in total.

In the following, we argue that in both cases that there exists an optimal solution where the items are placed exactly as the algorithm does assuming all the guesses are correct. When all the previous steps are correct, we can consider the residual set of items as a new instance which admits an optimal solution where all partially filled bins contain exactly two large items (and we already know the correct distribution of small items). For this new instance, we fill one bin correctly due to Claim 69 in the first case. Since this bin is filled correctly with respect to an existing optimal solution, we can again consider the residual set of items as an independent instance that needs solving. On the other hand, regarding the second case, we know by Claim 68 that there exists an optimal solution for this reduced instance where i_s is placed together with the largest fitting large item inside one partially filled bin. If we guess this bin correctly, we have filled one bin correctly with regard to the considered instance. Hence, when reducing the considered instance to the residual set of items (without this just filled bin), there exists an optimal solution for this instance with exactly one bin less.

After placing all the large items, we compare the obtained solution with the best solution so far, save it if it uses the smallest number of bins so far, and backtrack to the last decision. Since it iterates all possible guesses, this algorithm generates an optimal packing and its running time is bounded by $O((k!)^2 \cdot k \cdot 2^k \cdot N \log(N))$.

Everything that happens once can never happen again. But everything that happens twice will surely happen a third time.

— from *The Alchemist* by Paulo Coelho

11

ONLINE CARDINALITY CONSTRAINT SCHEDULING

The previous chapters present exact and approximation algorithms for various allocation problems. They all share the trait that the whole input is given at the beginning. However, this is too restrictive for some scenarios, as the input might change over time. This chapter focuses on scheduling problems where new jobs arrive. Consider for example a CPU scheduler, where clients regularly place new orders or further requirements have to be satisfied leading to new jobs.

Thus, we have to deal with uncertainty over time and aim for a sequence of good solutions throughout. Such algorithms, which operate on evolving instances, are called *online algorithms*. Their quality is measured by the *competitive ratio* which states the worst-case ratio between the solution of the respective online algorithm and an optimal offline solution, which knows the complete input.

In the *pure* online case, the algorithm is not allowed to make any changes in the current solution upon the arrival of new jobs. Thus, the goal is to place the (new) jobs such that the solution is as good as possible and extendable for further new jobs.

In some cases, we can relax this restriction. Then, the algorithm is allowed to modify the current solution. Clearly, modifications are costly and should be minimized. We measure the amount of allowed modification by the *migration factor*, which compares the total processing times of rescheduled jobs with the processing time of the new job. In particular, if we have a migration factor of β, we allow in round t (arriving of the tth job with processing time p_t) to reschedule a set of jobs $\mathcal{J}'$ with $\sum_{j \in \mathcal{J}'} p_j \leq \beta p_t$. Intuitively speaking, if a job with a small processing time arrives, we are only allowed to manipulate the solution a little. If, in turn, a job with a large processing time arrives, we can reschedule greater parts of the current solution.

It is known that there is a PTAS for the classical Scheduling problem on identical machines with a migration factor of $f(1/\epsilon)$ when jobs only arrive over time, i. e., an algorithm with competitive ratio of $(1+\epsilon)$ for some $\epsilon > 0$ [147]. This work was improved in [46] to obain an EPTAS and further extended in [156] to also hold when jobs depart. We call algorithms of the above kind *robust*, i. e., the migration factor directly corresponds to an improvement of the solution. Even though such robust algorithms exist for the case of identical machines, it seems hard to achieve similar results for the uniformly related ones. In fact, there is no algorithm with a competitive

ratio smaller than 2.564 for the pure online case on uniformly related machines [45]. The current best upper bound is an algorithm with a competitive ratio of 5.828 [17]. If migration is allowed, no further upper or lower bounds are known for this setting.

In the following, we want to explore the landscape for the CARDINALITY CONSTRAINT SCHEDULING problem in the online setting on identical and uniformly related machines. In this problem, we are given a set $\mathcal{J}$ of N jobs which arrive round-wise and each job j has a processing time p_j. Further, we have a set $\mathcal{M}$ of M machines, each machine i with a speed value s_i, and a positive integer c. The processing time of job j when scheduled on machine i is p_j/s_i. For identical machines, we have $s_i = 1$ for all machines. The objective is to find a schedule $\sigma : \mathcal{J} \to \mathcal{M}$ such that we place at most c jobs onto each machine and which minimizes the makespan $\mu(\sigma) = \max_{i\in\mathcal{M}} \sum_{j\in\sigma^{-1}(i)} p_j/s_i$. A generalization of this problem is the CLASS CONSTRAINT SCHEDULING problem where each job has some class and jobs from at most c classes can be scheduled on a machine. For an introduction to this problem and formal definitions, see Chapter 8. The cardinality constraint variant can be interpreted as the special case where each job has a unique class. Further, if we set $c = N$, both cases directly translate to the classical SCHEDULING problem and as such, we cannot hope to design better performing algorithms for these cases than we have bounds for the classical one. With this in mind, we aim to investigate the possibilities and limits of these constraint variants in the following.

In this regard, we show that the CLASS CONSTRAINT SCHEDULING problem is rather hopeless in the online setting. We construct for all migration factors β, which satisfy that we can still specify some $p_{\min}$ and $p_{\max}$ with $p_{\min} \leq 1/(\beta+1)p_{\max}$ (including the pure online case with $\beta = 0$), an example with a competitive ratio of M for identical machines. For uniformly related machines, this even yields an unbounded competitive ratio, i. e., a ratio dependent on the speed ratio between the fastest and the slowest machine. Note that β can be any constant or even depend on the values for M and c.

Turning our attention to the CARDINALITY CONSTRAINT SCHEDULING problem, we construct an example yielding a competitive ratio of at least $2 - 1/c$ for the pure online case on identical machines. Obviously, this result also holds for uniformly related machines, as it generalizes the identical ones. A general intuition supported by the aforementioned example is that we obtain a better ratio if we balance the jobs among the machines, i. e., at each point in time, every machine pair only differs by 1 regarding the number of jobs placed on them. Sadly, it is not that easy, as we show a competitive ratio of at least $M/4$ for this case on identical machines. Again, this directly transfers to the uniformly related machines. So, for the pure online case, already on identical machines, we cannot hope for something better than a constant competitive ratio. Further, the examples suggest that some potential algorithm to achieve this is probably rather complex.

On the bright side, if we consider c to be a constant or a parameter and allow $f(1/\epsilon, c)$ migration for some computable function f, we can design a robust EPTAS for the problem on identical machines, i. e., an algorithm

with running time $|I|^{O(1)} f_1(1/\epsilon, c)$ for some function f_1 and competitive/approximation ratio of $(1+\epsilon)$. The algorithm uses sensitivity results on the configuration ILP, i. e., upon the arrival of some job only few configurations and thus, machines change. The jobs corresponding to these configurations can then be rescheduled efficiently.

SUMMARY OF RESULTS

- We show a lower bound on the competitive ratio of M for the Class Constraint Scheduling problem on identical machines for any migration factor β such that we can construct some $p_{\min}$ and $p_{\max}$ with $p_{\min} \leq 1/(\beta+1)p_{\max}$. This includes the pure online case.
- We show a machine speed ratio dependent competitive ratio for the Class Constraint Scheduling problem on uniform machines for any migration factor β such that we can construct some $p_{\min}$ and $p_{\max}$ with $p_{\min} \leq 1/(\beta+1)p_{\max}$. This includes the pure online case.
- For the Cardinality Constraint Scheduling problem, we get a lower bound on the competitive ratio of $2 - 1/c$ for pure online case.
- If we restrict any algorithm to satisfy that at each point in time, every machine pair only differs by 1 regarding the number of jobs placed on them, we even obtain a lower bound of $M/4$ on the competitive ratio for the pure online case.
- For the Cardinality Constraint Scheduling problem on identical machines, we design a robust EPTAS with a migration factor of $f(1/\epsilon, c)$ for the case that c is a constant or a parameter.

FURTHER RELATED WORK For the classical Scheduling problem, there is a robust online EPTAS [156]. If we allow preemption of the jobs, i. e., we can split the jobs but not execute them in parallel, Epstein and Levin give an optimal online algorithm with $(1 - 1/M)$ migration [59]. We already discussed the few results for scheduling on uniformly related machines. If we consider unrelated machines, i. e., where each job j has an unrelated processing time $p_{j,i}$ on machine i, Azar et al. prove a lower bound on the competitive ratio of $\log^2(M)$ for the pure online setting [11]. Further, the aforementioned lower bounds for scheduling on identical machines with migration translate, as identical machines can be interpret as unrelated or uniformly related machines. Due to these deflating results, different migration approaches are studied. See for example [3, 50, 51] where the authors study the case of rescheduling k jobs after the last job arrived and study competitive ratios dependent on k. Further, in another setting, costs are assigned to the rescheduling of (certain) jobs and the goal is to minimize the total costs while obtaining a good schedule, see for example [10].

Turning back to our classical understanding of migration, in [156], the authors also studied the Santa Claus problem. There, the minimal load

on any machine should be maximized. The authors obtain a robust online algorithm for this problem. This result also holds for the case of departing jobs. On the other side, Skutella and Verschae prove in the same work [156] that there is no online algorithm for the Machine Covering problem with a competitive ratio of $20/19 - \epsilon$ for a migration factor of $f(1/\epsilon)$ for any function f.

Regarding the closely related Bin Packing problem, there is no algorithm that achieves a competitive ratio smaller than 1.54037 in the pure online case [13]. If, in turn, migration is allowed, Epstein and Levin give a robust APTAS, i. e., an algorithm with $(1+\epsilon)$ competitive ratio based on sensitivity results for ILPs [56]. This was improved by Jansen and Klein in [95] where they obtain the same ratio but reduce the worst-case migration to $\text{poly}(1/\epsilon)$. Berndt et al. extend this work to also handle the departure of items [18]. In the same work, the authors also prove that a migration factor of $\Omega(1/\epsilon)$ is needed to guarantee this competitive ratio.

Considering the cardinality constraint variant of Bin Packing, we refer to Chapter 8 for the offline variant. In the online setting, there is an algorithm with a competitive ratio of 2 [12, 16]. Further, in [13], Balogh et al. prove a lower bound of 2 for the overall competitive ratio if c converges to infinity, yielding that the current algorithms are tight. In the same work, they also prove bounds for specific values of c.

STRUCTURE OF THIS CHAPTER First, we consider the lower bounds on the class constraint variant in Section 11.1. In Section 11.2, we turn our attention to the Cardinality Constraint problem in the pure online setting. We present an example yielding a competitive ratio of at least $2 - 1/c$ and a competitive ratio of at least $M/4$ for the balanced case. Finally, we also investigate the other side of the coin in Section 11.3, where we present a robust EPTAS with migration factor $f(1/\epsilon, c)$ using configuration ILPs and sensitivity results.

11.1 LOWER BOUNDS FOR CLASS CONSTRAINT SCHEDULING

Let us start with the Class Constraint Scheduling problem and investigate its limits in the online setting. Recall that in this problem, we are given N jobs, each job j with a processing time p_j and class $c_j \in \{1, \dots, C\}$, and M machines with the same class restriction c. In contrast to the cardinality constraint variant, each class may have more than one job. Hence, the restriction for the machines does not limit the number of jobs scheduled on them but limits the number of classes the jobs belong to. We aim to find a schedule that minimizes the makespan. However, this seems rather hopeless in the online setting, even if we allow any migration factor β such that we can still construct some $p_{\min}$ and $p_{\max}$ with $p_{\min} \leq 1/(\beta+1)p_{\max}$: We show that the problem on identical machines admits a competitive ratio of M. For scheduling on uniformly related machines, the outlook is even worse. The ratio is unbounded, i. e., dependent on the speed ratio between the machines.

LOWER BOUND ON IDENTICAL MACHINES Let the migration factor be β, i. e., we allow in round t (arriving of the tth job with processing time p_t) to reschedule a set of jobs $\mathcal{J}'$ with $\sum_{j\in\mathcal{J}'} p_j \leq \beta p_t$.

For the competitive ratio of M, we start with receiving M jobs, each with processing time 1 and the same class c_1. They have to be placed onto the same machine i. Otherwise, if we place at least one job onto another machine, then for example $Mc-1$ jobs with classes $c_2, c_3, \ldots, c_{Mc}$ arrive, each with a processing time of $1/(\beta+1)$. As $1/(\beta+1)\cdot\beta < 1$, we cannot reschedule any of the first M jobs and thus, not sufficient class slots are available. Hence, we cannot construct a feasible schedule, even though there exists one in the offline setting when placing all jobs from the same class onto the same machine and c classes onto each one.

However, if we placed the jobs onto the same machine, the input ends after the first M jobs. It would have been optimal to place one job onto each machine to obtain the optimal makespan of 1. This yields a ratio of M.

LOWER BOUND ON UNIFORMLY RELATED MACHINES Let the migration factor be β. Assume that $c < M$. Let the speed values be $s_1 = 1$ for the first machine and $s > 1$ for the remaining $M-1$ ones, i. e., $s_2 = \cdots = s_M = s$. First, we receive Mc jobs, each with processing time 1 and a unique class $c_1, \ldots, c_{Mc}$. They have to be distributed among the machines such that each machine gets c different jobs. Without loss of generality, let us suppose that the first machine schedules the jobs from the classes $c_1, \ldots, c_c$. Next, we consider $\lceil c(\beta+1)\rceil$ rounds. In each round, c jobs arrive, each with a processing time smaller than or equal to $1/(\beta+1)$ and classes $c_1, \ldots, c_c$. In an optimal schedule, the classes $1, \ldots, c$ would be distributed among the c fastest machines. This would yield an optimal makespan of $c/s + c/s = 2c/s$. In the present schedule, they are all placed onto the slowest machine such that the makespan is $c + c^2$. Again, due to the small processing times, no rescheduling of the first jobs is possible. This yields a lower bound for the competitive ratio of $(c+c^2)/(2c/s) = (s+cs)/2 > cs$. This value might be arbitrarily large, as it also depends on the speed gap of the slowest machine to the largest one.

11.2 LOWER BOUNDS FOR CARDINALITY CONSTRAINT SCHEDULING WITHOUT MIGRATION

In the previous section, we show that the class constraint variant is rather hopeless in the online setting. In this section, we want to investigate the limits of the CARDINALITY CONSTRAINT SCHEDULING problem in the pure online setting, i. e., without migration. We show that we cannot hope for a better competitive ratio than $2 - 1/c$ by constructing an example yielding at least this ratio, already on identical machines. Following the general intuition, which is supported by that example, one might think that we obtain better ratios if we distribute the jobs evenly, i. e., at each point in time, the number of jobs on each machine pair differs by at most 1. However, we show that

an optimal approach is not that straight-forward. Indeed, if this property is required, we can even construct an example with a competitive ratio of at least $M/4$. Thus, an algorithm yielding a constant ratio, if existent, has to proceed more complex.

LOWER BOUND FOR THE GENERAL CASE Assume that $M \geq c-1$. We start with M rounds. In each round, $c-1$ jobs with processing times 1 arrive.

In the first case, the jobs are distributed evenly, i. e., we have $c-1$ jobs on each machine. Then, a huge job with processing time c arrives. We thus have a makespan of $c-1+c = 2c-1$. In an optimal schedule, however, we would have placed the $M(c-1)$ jobs onto $M-1$ machines. As $M \geq c-1$, the number of slots is sufficient. The last huge job would then be placed onto the empty machine yielding a makespan of c. Thus, the ratio is $(2c-1)/c = 2-1/c$.

In the other case, the jobs are not distributed evenly. Thus, there is at least one machine receiving at most $c-2$ jobs. Then, M huge jobs with processing time $\mathcal{N}$ arrive where $\mathcal{N}$ is some large number. In an optimal schedule, these jobs would be distributed evenly along the machines yielding a makespan of $\mathcal{N}+c-1$. However, due to the placing before, we have to place at least two of such jobs onto the same machine yielding a makespan of at least $2\mathcal{N}$. This yields a ratio of at least $(2\mathcal{N})/(\mathcal{N}+c-1)$, which is arbitrarily close to 2.

Overall, we thus get a competitive ratio of at least $2-1/c$. This value gets arbitrary close to 2 for large values of c (and $\mathcal{N}$ respectively). Note that identical machines are a special case of uniformly related machines. Thus, this lower bound also holds for them.

Further, note that the ratio in the second case gets worse the more class slots remain free on some machine. This hardens the intuition that an algorithm should try to balance the number of jobs each machine receives. However, this is not the full truth, as we show in the next example.

LOWER BOUND FOR THE BALANCED CASE Let ALG be an algorithm that maintains the invariant that the number of jobs placed on any two machines may differ by at most 1. In the following, we show that such an algorithm has a competitive ratio of at least $M/4$. Assume $c > 2^M M$.

Choose some machine i. Now, we consider $2^M M$ rounds, each with M jobs. Denote by ℓ_k the number of the job that is placed on i for each round $k = 1,\dots,2^M M$. The number is unique and between 1 and M due to the balancing constraint. Now, for each round k, we get a sequence of jobs such that the tth job with $t \leq \ell_k$ has a processing time 2^t, i. e., we get the processing times $2, 4, 8$ and so on. The remaining $M-\ell_k$ jobs arriving in the current round k have a processing time of 0.

Thus, in round k, machine i receives a load of 2^{ℓ_k}, where the overall load in that round is at most $2 \cdot 2^{\ell_k}$. Hence, machine i receives at least half of the overall load. The optimal makespan can be upper bounded by $p_{\max} + (\sum_{k=1}^{2^M M} 2 \cdot 2^{\ell_k})/M$, i. e., the maximal processing time among all jobs plus the average load. This can be seen, as we can simply schedule the jobs via round robin, a cyclic approach that places the jobs in non-ascending order

regarding their processing time onto the machines, which are repeatedly considered in some fixed order. This yields a makespan of at most $p_{\max} + (\sum_{k=1}^{2^M M} 2 \cdot 2^{\ell_k})/M$, see Lemma 41. Note that due to the magnitude of the rounds, $p_{\max} < (\sum_{k=1}^{2^M M} 2 \cdot 2^{\ell_k})/M$ as $p_{\max} \leq 2^M$ due to our definition of the processing times of each round.

Hence, we can estimate the ratio as follows:

$$
\begin{aligned}
&(\sum_{k=1}^{2^M M} 2^{\ell_k})/(p_{\max} + (\sum_{k=1}^{2^M M} 2 \cdot 2^{\ell_k})/M) \\
&\geq (\sum_{k=1}^{2^M M} 2^{\ell_k})/(2 \cdot (2 \sum_{k=1}^{2^M M} 2^{\ell_k})/M) \\
&= M/4.
\end{aligned}
$$

This example yields an overall competitive ratio of at least $M/4$ for this case. Note that identical machines are a special case of uniformly related machines. Thus, this lower bound also holds for them.

11.3 EPTAS FOR CARDINALITY CONSTRAINT SCHEDULING WITH MIGRATION

This section presents the positive results regarding online cardinality constraint scheduling if c is a constant or a parameter. In the following, we show a robust EPTAS with migration factor $f(1/\epsilon, c)$, i. e., an algorithm that admits a competitive ratio of $(1+\epsilon)$ and runs in time $|I|^{O(1)} f_1(1/\epsilon, c)$ for some computable function f_1, an accuracy parameter ϵ, the cardinality constraint c, and the encoding length $|I|$ of the instance I. The idea is to formulate the problem as a configuration ILP. Recall that a configuration is a multiplicity vector of processing times. The configuration ILP assigns one configuration onto each machine such that all jobs, i. e., the corresponding processing times, are covered.

When a new job arrives, the right-hand side of the Integer Linear Program (corresponding to the number of present processing times) only changes by one. Due to known sensitivity results, see for example Proposition 2, this implies that there exists an optimal solution for the new problem close to the old one. Thus, most of the configurations stay the same and hence, most jobs are placed as before. Setting up the configuration ILP for the few, unplaced jobs yields small ILP dimensions. Hence, the new ILP is solvable efficiently and a migration factor of $f(1/\epsilon, c)$ is obtained.

In the following, we first show the configuration ILP for this problem and then, present the complete algorithm. For a detailed introduction to sensitivity and configuration ILPs, we refer to the Preliminaries (Chapter 2).

The Configuration ILP

Denote the current round by t, i. e., the computation after receiving the tth job. Let the set of processing times present in round t be $\mathcal{P}^{(t)}$. Recall that a configuration $\kappa = (\kappa_1, \kappa_2, \ldots, \kappa_{|\mathcal{P}^{(t)}|})$ is a multiplicity vector of processing times stating that κ_j jobs with processing time $j \in \mathcal{P}^{(t)}$ are scheduled on the corresponding machine. We call a configuration $\kappa \in \mathcal{K}^{(t)}$ *valid* if $\sum_{j=1}^{|\mathcal{P}^{(t)}|} \kappa_j \leq c$ satisfying the cardinality constraints and $\sum_{j=1}^{|\mathcal{P}^{(t)}|} j\kappa_j \leq T$ fulfilling the makespan for some makespan guess T. Otherwise, the configuration is *non-valid*. Denote the set of valid configurations of round t as $\mathcal{K}^{(t)}$. Define the variable x_κ for the occurrence of the valid configuration κ. Denote by a_j the number of jobs with processing time j. The configuration ILP (config-ILP)$^{(t)}$ of round t looks as follows:

$$\sum_{\kappa \in \mathcal{K}^{(t)}} x_\kappa = M \tag{1}$$

$$\sum_{\kappa \in \mathcal{K}^{(t)}} x_\kappa \kappa_j = a_j \qquad \forall j = 1, \ldots, |\mathcal{P}^{(t)}| \tag{2}$$

The first constraint ensures that we use exactly one configuration for each machine. The second constraint is satisfied if all present jobs (processing times) are covered. It holds that (config-ILP)$^{(t)}$ has $|\mathcal{P}^{(t)}| + 1$ rows and $|\mathcal{K}^{(t)}|$ columns. We can bound the number of (valid) configurations roughly by $(c+1)^{|\mathcal{P}^{(t)}|}$, as each processing time can occur zero up to at most c times (in fact, the sum of all occurrences has to be lower than or equal to c). The largest number in the constraint matrix is $\kappa_j \leq c$ for some processing time j and configuration κ. The largest value for the right-hand side can be bounded by $\max\{M, \max_j\{a_j\}\} \leq t$, as we have t jobs in round t overall, and we set up the configuration ILP only if we have more jobs than machines as explained in the following.

The Algorithm

Place the first M jobs onto different machines. Afterwards, each time a new job j^* arrives, execute the following steps:

(1) compute a lower and upper bound on the optimal makespan;

(2) if the processing time of the new job j^* is smaller than δ/c times the current lower bound:

  (2.1) place it greedily onto a machine with less than c jobs.

(3) otherwise:

  (3.1) round the jobs;

  (3.2) interpret the previous schedule as a solution of (config-ILP)$^{(t)}$ and identify the jobs that remain untouched using sensitivity;

(3.3) set up and solve the reduced (config-ILP)$^{(t)}$ for the remaining jobs and assign them using the solution of the reduced (config-ILP)$^{(t)}$ onto the machines.

The step (3.3) requires that we know the desired (optimal) makespan. To circumvent this, we embed the last step in a binary search between the current lower and upper bound and take the lowest, feasible guess.

In the following, we go trough each step in more detail and directly argue their correctness to make sense of them. Let ϵ be the desired approximation ratio. Further, assume some accuracy value δ, which we specify in dependence of ϵ later on. Obviously, placing the first M jobs onto different machines is optimal. Thus, let us assume that we have placed at least M jobs, i. e., the next new job j^* is the tth job with $t \geq M+1$. The following steps are then executed.

Computing a lower and upper bound. An estimation for the lower bound $\mathrm{LB}^{(t)}$ of round t corresponds to an equal distribution of the overall processing time of all t present jobs and further, also considers the largest processing time $p^{(t)}_{\max}$ present in that round, i. e., $\mathrm{LB}^{(t)} = \max\{\lceil \sum_{j=1}^{t} p_j / M \rceil, p^{(t)}_{\max}\}$. As the makespan is always integral, we can safely round the value for the distributed load up to obtain an integer one. An upper bound for round t places the largest c jobs onto one machine, i. e., $\mathrm{UB}^{(t)} = c \cdot p^{(t)}_{\max}$. Note that the arrival of job j^* may change these bounds and thus, they have to be recalculated in each round.

Upon the arrival of a small job. If the processing time of the new job j^* is smaller than $\delta / c \, \mathrm{LB}^{(t)}$, we can place it greedily onto any machine which has less than c jobs. As j^* is relatively small, this only produces a small error. Otherwise, proceed with the following steps.

Rounding. For each processing time $p_j < \delta/c \cdot \mathrm{LB}^{(t)}$, set it to $p'_j = 0$. Otherwise, for all $p_j \geq \delta/c \cdot \mathrm{LB}^{(t)}$, round the processing times geometrically to $p'_j = \lceil (1+\delta)^\ell / \delta \rceil \cdot \delta^2/c \cdot \mathrm{LB}^{(t)}$ for some ℓ satisfying $\delta/c \cdot \mathrm{LB}^{(t)}(1+\delta)^{\ell-1} < p_j \leq \delta/c \cdot \mathrm{LB}^{(t)}(1+\delta)^\ell$. Denote the set of (different) rounded processing times of round t by $\mathcal{P}^{(t)}$. We can bound $|\mathcal{P}^{(t)}|$ by the least ℓ satisfying $\delta/c \cdot \mathrm{LB}(1+\delta)^\ell \geq p_{\max}$. It holds that

$$
\begin{aligned}
&\delta/c \cdot \mathrm{LB}(1+\delta)^\ell \geq p_{\max} \Rightarrow (1+\delta)^\ell \geq c/\delta \cdot p_{\max}/\mathrm{LB} \\
&\Rightarrow (1+\delta)^\ell \geq c/\delta \Rightarrow \ell \geq \log_{1+\delta}(c/\delta) \\
&\Rightarrow \ell \geq \log_e(c/\delta)/(\log_e(1+\delta)) \Rightarrow \ell \geq \log_e(c/\delta)/(\log_e(e^\delta)) \\
&\Rightarrow \ell \geq 1/\delta \log(c/\delta),
\end{aligned}
$$

as $p_{\max}/\mathrm{LB} \leq 1$ and $e^x \geq 1+x$ for all $x \in \mathbb{R}$. Thus, $|\mathcal{P}^{(t)}| \leq O(1/\delta \log(c/\delta))$. Further, we can rewrite the set of jobs more compactly as $(a_1, \ldots, a_{|\mathcal{P}^{(t)}|})$ where a_j states the number of jobs with processing time $j \in \mathcal{P}^{(t)}$.

Applying sensitivity. The schedule $\sigma^{(t-1)}$ of the previous round without inserting job j^* can be interpreted as a solution of $(\text{config-ILP})^{(t)}$ for the bounds and rounding of round t. To do so, first replace each job in the schedule $\sigma^{(t-1)}$ by its current rounded processing time in $\mathcal{P}^{(t)}$. Then, go through each machine and increase the value of x_κ for the corresponding configuration $\kappa \in \mathcal{K}^{(t)}$ by one. As we have no objective function, feasibility corresponds to optimality. Note that this solution might differ greatly from the solution of $(\text{config-ILP})^{(t-1)}$ as the jobs might be rounded differently. Nonetheless, the underlying schedule is the same and only changes slightly by the new job as we argue below.

Further, note that the configurations, which result from $\sigma^{(t-1)}$, i. e., from the schedule of the previous round, stay valid during the current round t: The current solution $\sigma^{(t-1)}$ yields a makespan of at most $(1 + O(\delta)) \cdot \mu(\text{OPT}(I^{(t-1)}))$ with respect to an optimal makespan $\mu(\text{OPT}(I^{(t-1)}))$ for the instance $I^{(t-1)}$ of round $t-1$, i. e., without job j^*, as proven in the next theorem. Introducing a new job cannot decrease the optimal makespan. Hence, $\mu(\text{OPT}(I^{(t-1)})) \leq \mu(\text{OPT}(I^{(t)}))$. Thus, we set the current makespan as the new lower bound (if it is larger than $\text{LB}^{(t)}$) introducing only a small error.

Due to the arrival of job j^*, the right-hand side only changes slightly: The corresponding a_j entry in the right-hand side is increased by one. Hence, we get that there exists an optimal solution (where at least the current set of configurations can be used) for the altered problem with distance

$$
\begin{aligned}
&|\mathcal{K}^{(t)}| \cdot c^{|\mathcal{P}^{(t)}|+1} \cdot (|\mathcal{P}^{(t)}|+1)^{(|\mathcal{P}^{(t)}|+1)/2} \\
&= (c+1)^{|\mathcal{P}^{(t)}|} \cdot c^{|\mathcal{P}^{(t)}|+1} \cdot (|\mathcal{P}^{(t)}|+1)^{(|\mathcal{P}^{(t)}|+1)/2} \\
&= c^{O(1/\delta \log(c/\delta))} \cdot (1/\delta \log(c/\delta))^{O(1/\delta \log(c/\delta))} \\
&= 2^{O(1/\delta \log^2(c/\delta))}
\end{aligned}
$$

due to the sensitivity bound and the Hadamard inequality, see Proposition 2 and Chapter 2. Thus, we can leave $x'_\kappa = \max\{\lceil x_\kappa - 2^{O(1/\delta \log^2(c/\delta))} \rceil, 0\}$ many configuration κ untouched for each $\kappa \in \mathcal{K}^{(t)}$. Hence, we are left with at most $(c+1)^{|\mathcal{P}^{(t)}|} \cdot 2^{O(1/\delta \log^2(c/\delta))} \cdot c = 2^{O(1/\delta \log^2(c/\delta))}$ unassigned jobs (for each configuration, appearing sensitivity often, we get at most c jobs) where j^* is inserted to.

Scheduling the remaining jobs. We have to pack the remaining $2^{O(1/\delta \log^2(c/\delta))}$ jobs onto $M' = M - \sum_{\kappa \in \mathcal{K}^{(t)}} x'_\kappa$ left over machines. As before, we can interpret them as a vector $(a'_1, \ldots, a'_{|\mathcal{P}^{(t)}|})$. To set up the reduced configuration ILP $(\text{config-ILP})^{(t)}$ of round t, we have to know the desired makespan, as this guess influences the set of valid configurations. Thus, we proceed with a binary search between the lower bound $\text{LB}^{(t)}$ and upper bound $\text{UB}^{(t)}$ of the current round and take the lowest guess for which the reduced $(\text{config-ILP})^{(t)}$ admits a solution (for example using the algorithm in [104]). Having this solution at hand, we can place the configurations (processing time placeholders) onto the remaining machines according to the x_κ variables. Then, we fill

the placeholder greedily with jobs admitting the corresponding processing time. Finally, we reinsert the original processing times for all jobs.

Theorem 70. *The algorithm above computes at each round t a schedule $\sigma^{(t)}$ for the online variant of the* Cardinality Constrained Scheduling *problem with makespan $\mu(\sigma^{(t)}) \leq (1+\epsilon)\mu(\text{OPT}(I^{(t)}))$ and migration factor $2^{O(1/\epsilon \log^2(c/\epsilon))}$ in time $O(t) + \log(p_{\max}^{(t)}) 2^{O(1/\epsilon \log^2(c/\epsilon))}$ where $\text{OPT}(I^{(t)})$ is a solution with optimal makespan for scheduling the instance $I^{(t)}$. This yields the desired, robust EPTAS for this problem.*

Proof. We already argued the correctness of the algorithm to make sense of the steps. It remains to argue the migration factor and error. Further, we have to analyze the running time.

In the first M rounds, we place the jobs greedily onto free machines. Thus, we do not migrate any jobs. The same holds when a small job with processing time smaller than $\delta/c\,\text{LB}^{(t)}$ arrives, as we place it greedily onto a machine which still can schedule another job. By that, the error is bounded by $O(\delta)$ due to the small processing time.

Otherwise, as the current job then admits a processing time of at least $\delta/c\,\text{LB}^{(t)}$, we are allowed to reschedule a total load of

$$\begin{aligned}
&2^{O(1/\delta \log^2(c/\delta))} \cdot \delta/c \cdot \text{LB}^{(t)} \\
&= 2^{O(1/\delta \log^2(c/\delta))} \cdot 2^{\log(\delta/c)} \cdot \text{LB}^{(t)} \\
&= 2^{O(1/\delta \log^2(c/\delta)) + \log(\delta/c)} \cdot \text{LB}^{(t)} \\
&\geq 2^{O(1/\delta \log^2(c/\delta))} \cdot \text{LB}^{(t)} \\
&\geq 2^{O(1/\delta \log^2(c/\delta))} \cdot p_{\max}^{(t)}.
\end{aligned}$$

We reschedule $2^{O(1/\delta \log^2(c/\delta))}$ many jobs due to the sensitivity, each with a processing time of at most $p_{\max}^{(t)}$ yielding the desired migration factor. The inaccuracy we introduce in that case is by rounding and assuming the schedule of the last round to be optimal. Regarding the rounding of round t, for processing times with $p_j < \delta/c\,\text{LB}^{(t)}$, we set them to zero. As at most c of them can be placed onto one machine, this yields an error of δ.

Similar, by rounding the remaining processing times $p_j \geq \delta/c$ LB, we extend them by a factor of at most $(1+O(\delta))$ as

$$\begin{aligned}
&\lceil (1+\delta)^{\ell}/\delta \rceil \cdot \delta^2/c \cdot \text{LB} \\
&\leq ((1+\delta)^{\ell}/\delta + 1) \cdot \delta^2/c \cdot \text{LB} \\
&= (1+\delta)^{\ell} \delta/c\,\text{LB} + \delta^2/c \cdot \text{LB} \\
&\leq (1+\delta)p_j + \delta p_j = (1+O(\delta))p_j
\end{aligned}$$

using that $p_j \geq \delta/c \cdot \text{LB}\,(1+\delta)^{\ell-1}$ by the rounding and $p_j \geq \delta/c\,\text{LB}$ per definition. Thus, the complete schedule is enlarged by at most the same factor yielding an error of $O(\delta)$. The previous schedule admits a makespan of at most $(1+O(\delta)) \cdot \mu(\text{OPT}(I^{(t-1)}))$ with respect to an optimal makespan

$\mu(\text{OPT}(I^{(t-1)}))$ for the instance $I^{(t-1)}$ of round $t-1$, i. e., without job j^*. Thus, by setting this makespan as the new lower bound, we produce an error of $O(\delta)$. Overall, this leads to an error of $O(\delta)$. Thus, setting $\epsilon = O(\delta)$ yields the desired ratio.

Regarding the running time, it takes time $O(t)$ to compute the bounds and round the instance, as we have to go trough each processing time once and t jobs are present. Applying the sensitivity results to schedule parts of the instance also takes time $O(t)$, as we go trough each job (machine-wise) to get the current set of chosen configurations, i. e., the values for the variables x_κ. For the reduced (config-ILP)$^{(t)}$, the new value for the right-hand side is bounded by $2^{O(1/\delta \log^2(c/\delta))}$ (maximal number of remaining jobs), the other parameters stay as estimated above. Using the algorithm by Jansen and Rohwedder [104], it thus takes time $(|\mathcal{P}^{(t)}|c)^{O(|\mathcal{P}^{(t)}|)} \log(2^{O(1/\delta \log^2(c/\delta))}) + O(2^{O(1/\delta \log^2(c/\delta))}) = 2^{O(1/\delta \log^2(c/\delta))}$ to solve the reduced (config-ILP)$^{(t)}$. This is embedded in a binary search with at most $\log(\text{UB}^{(t)}) = \log(cp_{\max}^{(t)})$ iterations. Assigning the configurations, the jobs and finally the original processing times is again possible in linear time. Thus, we get an overall running time of

$$\begin{aligned} &O(t) + \log(cp_{\max}^{(t)}) 2^{O(1/\delta \log^2(c/\delta))} \\ &= O(t) + \log(p_{\max}^{(t)}) 2^{O(1/\epsilon \log^2(c/\epsilon))} \end{aligned}$$

finishing the proof. □

A dozen more questions occurred to me. Not to mention twenty-two possible solutions to each one, sixteen resulting hypotheses and counter-theorems, eight abstract speculations, a quadrilateral equation, two axioms, and a limerick.

— from *Ptolemy's Gate (Bartimaeus Trilogy)* by Jonathan Stroud

12 OPEN QUESTIONS

This part considered various allocation problems such as scheduling with clique incompatibilities, scheduling with class restrictions in the offline and online setting, and vector variants of Bin Packing and Multiple Knapsack. The main idea is to employ the well-structured ILPs and their structural properties introduced in Part I to design efficient algorithms which produce satisfactory good solutions. In doing so, we reaffirm the usefulness of these integer linear programs to deal with allocation problems and the fruitful symbiosis of approximation algorithms and FPT approaches.

Still, even though the present chapters already show quite a number of results, many interesting research directions are still open. Regarding clique incompatibilities, for instance, we are quite interested in a more detailed study of our setting from the perspective of (FPT) approximation algorithms. This includes approximation algorithms with FPT running times. The most obvious question in this context probably is whether a constant rate approximation for $P|\text{cliques}, M(k)|\sum C_j$ is possible, given that this problem is APX-hard. Further, the study of different sensible classes of incompatibility graphs for the total completion time objective seems worthwhile.

For the Variable Class Constraint Scheduling problem, we developed a PTAS for each variant of feasibly allotting the jobs, i. e., for the non-preemptive, the splittable and the preemptively case. However, it still remains open if an EPTAS is possible. If this question is also tackled by formulating appropriate block-structured ILPs, the main obstacle is to circumvent that the number of class slots on a machine (c in the identical case and k_i for machine i otherwise) is a factor in the constraint matrix. But, of course, a solution is not allowed to violate the class restrictions. Thus, it has to be taken care of at some other point.

Further, we broaden the scope of our conference paper [100] by allowing different machine types. However, we have to parameterize over the number of different ones to obtain the PTAS results. This raises the questions whether we can design (E)PTASs for the case where the number of machine types is not a parameter. The current bottleneck which keeps us from doing so is the iteration through some set that states the number of remaining class slots after placing large jobs. This set can have up to machine type many, different numbers, which directly influences the number r of globally uniform constraints.

Regarding the algorithms for the vector variants of Bin Packing and Multiple Knapsack, a crucial and also time consuming step is to guess the form of the packing regarding the small items and the bins containing only small items, one or respectively two additional large items. It would be favorable to circumvent this costly guessing by integrating this step into the matching problem, for example by exploring algebraic methods. Recently, these methods achieved great attention in the FPT community, see for example [77]. Roughly speaking, the goal is to associate a polynomial whose terms enumerate the sets of small items that would fit in the corresponding bin. Then, techniques such as algebraic sieving could be used to look for a matching that covers all items.

Finally, let us turn our attention to the online setting of the Class Constraint Scheduling problem. We have seen that this problem is rather hopeless even if we allow migration. However, studying amortized migration may yield better competitive ratios. Roughly speaking, amortized migration allows to re-schedule jobs in dependence of the migration factor and the total load of new jobs since the last repacking, not only the load of the current new job. Thus, we can save up to re-schedule more jobs later on. This approach might also lead to better online algorithms regarding the competitive ratio for the Cardinality Constraint Scheduling problem.

Finally, we want to mention that upper and lower bounds for the classical Scheduling problem in the online setting on uniformly related machines is still an interesting research direction with only few results so far.

BIBLIOGRAPHY

[1] M. Aigner and G. M. Ziegler. *Proofs from THE BOOK*. Springer, 2018.

[2] J. Akiyama and M. Kano. "Matchings and 1-Factors." In: *Factors and Factorizations of Graphs*. Springer, 2011, pp. 15–67.

[3] S. Albers and M. Hellwig. "On the value of job migration in online makespan minimization." In: *Algorithmica* 79.2 (2017), pp. 598–623.

[4] I. Aliev, M. Henk, and T. Oertel. "Distances to lattice points in knapsack polyhedra." In: *Mathematical Programming* (2019), pp. 1–24.

[5] N. Alon, Y. Azar, J. Csirik, L. Epstein, S. V. Sevastianov, A. P. A. Vestjens, and G. J. Woeginger. "On-line and off-line approximation algorithms for vector covering problems." In: *Algorithmica* 21.1 (1998), pp. 104–118.

[6] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. "Approximation schemes for scheduling on parallel machines." In: *Journal of Scheduling* 1.1 (1998), pp. 55–66.

[7] N. Alon, R. Yuster, and U. Zwick. "Color-Coding." In: *Journal of the ACM* 42.4 (1995), pp. 844–856.

[8] K. Altmanová, D. Knop, and M. Koutecký. "Evaluating and Tuning n-fold Integer Programming." In: *ACM Journal of Experimental Algorithmics* 24.1 (2019), 2.2:1–2.2:22.

[9] A. Andoni and K. Onak. "Approximating Edit Distance in Near-Linear Time." In: *SIAM Journal of Computing* 41.6 (2012), pp. 1635–1648.

[10] M. Andrews, M. X. Goemans, and L. Zhang. "Improved Bounds for On-Line Load Balancing." In: *Algorithmica* 23.4 (1999), pp. 278–301.

[11] Y. Azar, J. Naor, and R. Rom. "The Competitiveness of On-Line Assignments." In: *Journal of Algorithms* 18.2 (1995), pp. 221–237.

[12] L. Babel, B. Chen, H. Kellerer, and V. Kotov. "Algorithms for on-line bin-packing problems with cardinality constraints." In: *Discrete Applied Mathematics* 143.1-3 (2004), pp. 238–251.

[13] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. "Online bin packing with cardinality constraints resolved." In: *Journal of Computer and System Sciences* 112 (2020), pp. 34–49.

[14] M. Bannach, S. Berndt, M. Maack, M. Mnich, A. Lassota, M. Rau, and M. Skambath. "Solving Packing Problems with Few Small Items Using Rainbow Matchings." In: *MFCS*. Vol. 170. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 11:1–11:14.

[15] N. Bansal, M. Eliás, and A. Khan. "Improved approximation for vector bin packing." In: *SODA*. 2016, pp. 1561–1579.

[16] J. Békési, G. Dósa, and L. Epstein. "Bounds for online bin packing with cardinality constraints." In: *Information and Computation* 249 (2016), pp. 190–204.

[17] P. Berman, M. Charikar, and M. Karpinski. "On-Line Load Balancing for Related Machines." In: *Journal of Algorithms* 35.1 (2000), pp. 108–121.

[18] S. Berndt, K. Jansen, and K. Klein. "Fully dynamic bin packing revisited." In: *Mathematical Programming* 179.1 (2020), pp. 109–155.

[19] S. Berndt, K. Jansen, and A. Lassota. "Tightness of Sensitivity and Proximity Bounds for Integer Linear Programs." In: *SOFSEM*. Vol. 12607. Lecture Notes in Computer Science. Springer, 2021, pp. 349–360.

[20] R. van Bevern. "Towards optimal and expressive kernelization for d-hitting set." In: *Algorithmica* 70.1 (2014), pp. 129–147.

[21] H. L. Bodlaender and K. Jansen. "On the Complexity of Scheduling Incompatible Jobs with Unit-Times." In: *MFCS*. Vol. 711. Lecture Notes in Computer Science. Springer, 1993, pp. 291–300.

[22] H. L. Bodlaender, K. Jansen, and G. J. Woeginger. "Scheduling with Incompatible Jobs." In: *Discrete Applied Mathematics* 55.3 (1994), pp. 219–232.

[23] C. Brand, M. Koutecký, and S. Ordyniak. "Parameterized Algorithms for MILPs with Small Treedepth." In: *CoRR* abs/1912.03501 (2019).

[24] P. Brucker. *Scheduling algorithms (4. edition)*. Springer, 2004.

[25] M. Bruglieri, M. Ehrgott, H. W. Hamacher, and F. Maffioli. "An annotated bibliography of combinatorial optimization problems with fixed cardinality constraints." In: *Discrete Applied Mathematics* 154.9 (2006), pp. 1344–1357.

[26] J. L. Bruno, E. G. C. Jr., and R. Sethi. "Scheduling Independent Tasks to Reduce Mean Finishing Time." In: *Communications of the ACM* 17.7 (1974), pp. 382–387.

[27] A. Caprara, M. Dell'Amico, J. C. D. Díaz, M. Iori, and R. Rizzi. "Friendly bin packing instances without Integer Round-up Property." In: *Mathematical Programming* 150.1 (2015), pp. 5–17.

[28] A. Caprara, H. Kellerer, and U. Pferschy. "Approximation schemes for ordered vector packing problems." In: *Naval Research Logistics* 50.1 (2003), pp. 58–69.

[29] B. Chen, C. N. Potts, and G. J. Woeginger. "A review of machine scheduling: Complexity, algorithms and approximability." In: *Handbook of combinatorial optimization*. Springer, 1998, pp. 1493–1641.

[30] L. Chen, K. Jansen, W. Luo, and G. Zhang. "An Efficient PTAS for Parallel Machine Scheduling with Capacity Constraints." In: *COCOA*. 2016, pp. 608–623.

[31] L. Chen, M. Koutecký, L. Xu, and W. Shi. "New Bounds on Augmenting Steps of Block-Structured Integer Programs." In: *ESA*. Vol. 173. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 33:1–33:19.

[32] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali. "Approximation and online algorithms for multidimensional bin packing: A survey." In: *Computer Science Review* 24 (2017), pp. 63–79.

[33] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory Of Scheduling*. Adison-Wesly, 1967.

[34] W. J. Cook, J. Fonlupt, and A. Schrijver. "An integer analogue of Carathéodory's theorem." In: *Journal of Combinatorial Theory, Series B* 40.1 (1986), pp. 63–70.

[35] W. J. Cook, A. M. H. Gerards, A. Schrijver, and É. Tardos. "Sensitivity theorems in integer linear programming." In: *Mathematical Programming* 34.3 (1986), pp. 251–264.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[37] R. Crandall and C. B. Pomerance. *Prime numbers: a computational perspective*. Vol. 182. Springer Science & Business Media, 2006.

[38] J. Csirik, J. B. G. Frenk, M. Labbé, and S. Zhang. "On the multidimensional vector bin packing." In: *Acta Cybernetica* 9.4 (1990), pp. 361–369.

[39] J. Cslovjecsek, F. Eisenbrand, C. Hunkenschröder, L. Rohwedder, and R. Weismantel. "Block-Structured Integer and Linear Programming in Strongly Polynomial and Near Linear Time." In: *SODA*. SIAM, 2021, pp. 1666–1681.

[40] J. Cslovjecsek, F. Eisenbrand, M. Pilipczuk, M. Venzin, and R. Weismantel. "Efficient sequential and parallel algorithms for multistage stochastic integer programming using proximity." In: *CoRR* abs/2012.11742 (2020).

[41] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[42] S. Das and A. Wiese. "On Minimizing the Makespan When Some Jobs Cannot Be Assigned on the Same Machine." In: *ESA*. Vol. 87. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 31:1–31:14.

[43] T. Dokka, A. Kouvela, and F. C. R. Spieksma. "Approximating the multi-level bottleneck assignment problem." In: *Operations Research Letters* 40.4 (2012), pp. 282–286.

[44] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[45] T. Ebenlendr and J. Sgall. "A Lower Bound on Deterministic Online Algorithms for Scheduling on Related Machines Without Preemption." In: *Theory of Computing Systems* 56.1 (2015), pp. 73–81.

[46] T. Ehlers and K. Jansen. "Online-Scheduling on Identical Machines with Bounded Migration." In: *SYNASC*. IEEE Computer Society, 2013, pp. 361–366.

[47] F. Eisenbrand, C. Hunkenschröder, and K. Klein. "Faster Algorithms for Integer Programs with Block Structure." In: *ICALP*. Vol. 107. LIPIcs. 2018, 49:1–49:13.

[48] F. Eisenbrand, C. Hunkenschröder, K. Klein, M. Koutecký, A. Levin, and S. Onn. "An Algorithmic Theory of Integer Programming." In: *CoRR* abs/1904.01361 (2019).

[49] F. Eisenbrand and R. Weismantel. "Proximity Results and Faster Algorithms for Integer Programming Using the Steinitz Lemma." In: *ACM Transactions on Algorithms* 16.1 (2020), 5:1–5:14.

[50] M. Englert, D. Mezlaf, and M. Westermann. "Online Makespan Scheduling with Job Migration on Uniform Machines." In: *ESA*. Vol. 112. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 26:1–26:14.

[51] M. Englert, D. Özmen, and M. Westermann. "The Power of Reordering for Online Minimum Makespan Scheduling." In: *SIAM Journal on Computing* 43.3 (2014), pp. 1220–1237.

[52] L. Epstein, L. M. Favrholdt, and A. Levin. "Online variable-sized bin packing with conflicts." In: *Discrete Optimization* 8.2 (2011), pp. 333–343.

[53] L. Epstein, A. Levin, and R. van Stee. "Approximation schemes for packing splittable items with cardinality constraints." In: *Algorithmica* 62.1-2 (2012), pp. 102–129.

[54] L. Epstein, C. Imreh, and A. Levin. "Class constrained bin packing revisited." In: *Theoretical Computer Science* 411.34-36 (2010), pp. 3073–3089.

[55] L. Epstein, K. Jansen, A. Lassota, A. Levin, M. Maack, and L. Rohwedder. *Cardinality Constraint Scheduling: Pure Online and Migration*. (in progress). 2021.

[56] L. Epstein and A. Levin. "A robust APTAS for the classical bin packing problem." In: *Mathematical Programming* 119.1 (2009), pp. 33–49.

[57] L. Epstein and A. Levin. "AFPTAS results for common variants of bin packing: A new method for handling the small items." In: *SIAM Journal on Optimization* 20.6 (2010), pp. 3121–3145.

[58] L. Epstein and A. Levin. "Robust Approximation Schemes for Cube Packing." In: *SIAM Journal of Optimization* 23.2 (2013), pp. 1310–1343.

[59] L. Epstein and A. Levin. "Robust Algorithms for Preemptive Scheduling." In: *Algorithmica* 69.1 (2014), pp. 26–57.

[60] S. Fafianie and S. Kratsch. "A shortcut to (sun)flowers: kernels in logarithmic space or linear time." In: *MFCS*. Vol. 9235. Lecture Notes in Computer Science. 2015.

[61] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[62] M. L. Fredman, J. Komlós, and E. Szemerédi. "Storing a Sparse Table with $O(1)$ Worst Case Access Time." In: *Journal of the ACM* 31.3 (1984), pp. 538–544.

[63] H. Furmańczyk and M. Kubale. "Scheduling of unit-length jobs with bipartite incompatibility graphs on four uniform machines." In: *Bulletin of the Polish Academy of Sciences: Technical Sciences* 65 (2017), pp. 29–34.

[64] H. Furmańczyk and M. Kubale. "Scheduling of unit-length jobs with cubic incompatibility graphs on three uniform machines." In: *Discrete Applied Mathematics* 234 (2018), pp. 210–217.

[65] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao. "Resource constrained scheduling as generalized bin packing." In: *Journal of Combinatorial Theory, Series A* 21.3 (1976), pp. 257–298.

[66] M. R. Garey and D. S. Johnson. "Approximation algorithms for bin packing problems: A survey." In: *Analysis and design of algorithms in combinatorial optimization*. Springer, 1981.

[67] T. Gavenčiak, M. Koutecký, and D. Knop. "Integer programming in parameterized complexity: Five miniatures." In: *Discrete Optimization* (2020), p. 100596.

[68] S. Geng and L. Zhang. "The complexity of the 0/1 multi-knapsack problem." In: *Journal of Computer Science and Technology* 1.1 (1986), pp. 46–50.

[69] P. C. Gilmore and R. E. Gomory. "A linear programming approach to the cutting-stock problem." In: *Operations research* 9.6 (1961), pp. 849–859.

[70] M. X. Goemans and T. Rothvoß. "Polynomiality for Bin Packing with a Constant Number of Item Types." In: *SODA*. SIAM, 2014, pp. 830–839.

[71] M. X. Goemans and D. P. Williamson. "Two-Dimensional Gantt Charts and a Scheduling Algorithm of Lawler." In: *SIAM Journal on Discrete Mathematics* 13.3 (2000), pp. 281–294.

[72] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. "Approximation algorithms for data placement on parallel disks." In: *ACM Transactions on Algorithms* 5.4 (2009), p. 34.

[73] T. F. Gonzalez. *Handbook of approximation algorithms and metaheuristics*. Chapman and Hall/CRC, 2007.

[74] K. Grage, K. Jansen, and K. Klein. "An EPTAS for Machine Scheduling with Bag-Constraints." In: *SPAA*. ACM, 2019, pp. 135–144.

[75] R. L. Graham. "Bounds on Multiprocessing Timing Anomalies." In: *SIAM Journal of Applied Mathematics* 17.2 (1969), pp. 416–429.

[76] J. Guo, F. Hüffner, and R. Niedermeier. "A structural view on parameterizing problems: distance from triviality." In: *IWPEC*. Vol. 3162. Lecture Notes in Computer Science. 2004, pp. 162–173.

[77] G. Z. Gutin, D. Majumdar, S. Ordyniak, and M. Wahlström. "Parameterized Pre-Coloring Extension and List Coloring Problems." In: *STACS*. Vol. 154. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 19:1–19:18.

[78] G. Z. Gutin, M. Wahlström, and A. Yeo. "Rural postman parameterized by the number of components of required edges." In: *Journal of Computer and System Sciences* 83.1 (2017), pp. 121–131.

[79] J. Hadamard. "Resolution d'une question relative aux determinants." In: *Bulletin des Sciences Mathématiques* 2 (1893), pp. 240–246.

[80] R. W. Haessler and P. E. Sweeney. "Cutting stock problems and solution procedures." In: *European Journal of Operational Research* 54.2 (1991), pp. 141–150.

[81] G. H. Hardy, J. E. Littlewood, et al. "Contributions to the theory of the Riemann zeta-function and the theory of the distribution of primes." In: *Acta Mathematica* 41 (1916), pp. 119–196.

[82] R. Hemmecke, M. Köppe, and R. Weismantel. "A Polynomial-Time Algorithm for Optimizing over N-Fold 4-Block Decomposable Integer Programs." In: *IPCO*. Vol. 6080. Lecture Notes in Computer Science. Springer, 2010, pp. 219–229.

[83] R. Hemmecke, S. Onn, and L. Romanchuk. "n-Fold integer programming in cubic time." In: *Mathematical Programming* 137.1-2 (2013), pp. 325–341.

[84] R. Hemmecke, S. Onn, and R. Weismantel. "n-fold integer programming and nonlinear multi-transshipment." In: *Optimization Letters* 5.1 (2011), pp. 13–25.

[85] R. Hemmecke and R. Schultz. "Decomposition of test sets in stochastic integer programming." In: *Mathematical Programming* 94.2-3 (2003), pp. 323–341.

[86] R. Hoberg and T. Rothvoß. "A logarithmic additive integrality gap for bin packing." In: *SODA*. 2017, pp. 2616–2625.

[87] D. S. Hochbaum. "Monotonizing linear programs with up to two nonzeroes per column." In: *Operations Research Letters* 32.1 (2004), pp. 49–58.

[88] D. S. Hochbaum and J. G. Shanthikumar. “Convex Separable Optimization Is Not Much Harder than Linear Optimization.” In: *ACM* 37.4 (1990), pp. 843–862.

[89] D. S. Hochbaum and D. B. Shmoys. “Using dual approximation algorithms for scheduling problems theoretical and practical results.” In: *Journal of the ACM* 34.1 (1987), pp. 144–162.

[90] D. A. Holton and J. Sheehan. *The Petersen Graph*. Vol. 7. Cambridge University Press, 1993.

[91] R. Impagliazzo and R. Paturi. “On the Complexity of k-SAT.” In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 367–375.

[92] R. Impagliazzo, R. Paturi, and F. Zane. “Which Problems Have Strongly Exponential Complexity?” In: *Journal of Computer and System Sciences* 63.4 (2001), pp. 512–530.

[93] K. Ireland and M. Rosen. *A classical introduction to modern number theory*. Vol. 84. Graduate texts in mathematics. Springer, 1982.

[94] K. Jansen and K.-M. Klein. “About the structure of the integer cone and its application to bin packing.” In: *SODA*. 2017, pp. 1571–1581.

[95] K. Jansen and K. Klein. “A Robust AFPTAS for Online Bin Packing with Polynomial Migration.” In: *SIAM Journal on Discrete Mathematics* 33.4 (2019), pp. 2062–2091.

[96] K. Jansen, K. Klein, and A. Lassota. “The Double Exponential Runtime is Tight for 2-Stage Stochastic ILPs.” In: *IPCO*. Lecture Notes in Computer Science. to appear. Springer, 2021.

[97] K. Jansen, K. Klein, M. Maack, and M. Rau. “Empowering the Configuration-IP - New PTAS Results for Scheduling with Setups Times.” In: *ITCS*. Vol. 124. LIPIcs. 2019, 44:1–44:19.

[98] K. Jansen, K. Klein, and J. Verschae. “Closing the Gap for Makespan Scheduling via Sparsification Techniques.” In: *ICALP*. Vol. 55. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 72:1–72:13.

[99] K. Jansen, S. Kratsch, D. Marx, and I. Schlotter. “Bin packing with fixed number of bins revisited.” In: *Journal of Computer and System Sciences* 79.1 (2013), pp. 39–49.

[100] K. Jansen, A. Lassota, and M. Maack. “Approximation Algorithms for Scheduling with Class Constraints.” In: *SPAA*. ACM, 2020, pp. 349–357.

[101] K. Jansen, A. Lassota, M. Maack, and T. Pikies. “Total Completion Time Minimization for Scheduling with Incompatibility Cliques.” In: *ICAPS*. to appear. AAAI Press, 2021.

[102] K. Jansen, A. Lassota, and L. Rohwedder. "Near-Linear Time Algorithm for n-fold ILPs via Color Coding." In: *SIAM Journal on Discrete Mathematics* 34.4 (2020). An extended abstract of this work appeared in the proceedings of the 46th International Colloquium on Automata, Languages and Programming (ICALP 2019), pp. 2282–2299.

[103] K. Jansen and C. Robenek. "Scheduling Jobs on Identical and Uniform Processors Revisited." In: *WAOA*. Vol. 7164. Lecture Notes in Computer Science. Springer, 2011, pp. 109–122.

[104] K. Jansen and L. Rohwedder. "On Integer Programming and Convolution." In: *ITCS*. Vol. 124. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 43:1–43:17.

[105] K. Jansen and R. Solis-Oba. "A polynomial time *OPT* + 1 algorithm for the cutting stock problem with a constant number of object lengths." In: *Mathematics of Operations Research* 36.4 (2011), pp. 743–753.

[106] H. W. L. Jr. "Integer Programming with a Fixed Number of Variables." In: *Mathematics of Operations Research* 8.4 (1983), pp. 538–548.

[107] P. Kall and S. W. Wallace. *Stochastic programming*. Springer, 1994.

[108] R. Kannan. "Minkowski's Convex Body Theorem and Integer Programming." In: *Mathematics of Operations Research* 12.3 (1987), pp. 415–440.

[109] M. Kano and X. Li. "Monochromatic and heterochromatic subgraphs in edge-colored graphs–A survey." In: *Graphs and Combinatorics* 24.4 (2008), pp. 237–263.

[110] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[111] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford. "An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations." In: *SODA*. 2014, pp. 217–226.

[112] A. B. Kempe. "A memoir in the theory of mathematical form." In: *Philosophical Transactions of the Royal Society London* 177 (1886), pp. 1–70.

[113] C. Kenyon. "Best-fit bin-packing with random order." In: *SODA*. 1996, pp. 359–364.

[114] K. Klein. "About the Complexity of Two-Stage Stochastic IPs." In: *IPCO*. Vol. 12125. Lecture Notes in Computer Science. Springer, 2020, pp. 252–265.

[115] D. Knop and M. Koutecký. "Scheduling meets n-fold integer programming." In: *Journal of Scheduling* 21.5 (2018), pp. 493–503.

[116] D. Knop, M. Koutecký, A. Levin, M. Mnich, and S. Onn. "Multitype Integer Monoid Optimization and Applications." In: *CoRR* abs/1909.07326 (2019).

[117] D. Knop, M. Koutecký, and M. Mnich. "Combinatorial n-fold Integer Programming and Applications." In: *ESA*. Vol. 87. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 54:1–54:14.

[118] D. Knop, M. Pilipczuk, and M. Wrochna. "Tight Complexity Lower Bounds for Integer Linear Programming with Few Constraints." In: *STACS*. Vol. 126. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 44:1–44:15.

[119] M. Koutecký, A. Levin, and S. Onn. "A Parameterized Strongly Polynomial Algorithm for Block Structured Integer Programs." In: *ICALP*. Vol. 107. LIPIcs. 2018, 85:1–85:14.

[120] S. Küçükyavuz and S. Sen. "An introduction to two-stage stochastic mixed-integer programming." In: *Leading Developments from INFORMS Communities*. INFORMS, 2017, pp. 1–27.

[121] J. Kuipers. "Bin packing games." In: *Mathematical Methods of Operations Research* 47.3 (1998), pp. 499–510.

[122] G. Lamé. *Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers*. 1844.

[123] V. B. Le and F. Pfender. "Complexity results for rainbow matchings." In: *Theoretical Computer Science* 524 (2014), pp. 27–33.

[124] J. Lee, J. Paat, I. Stallknecht, and L. Xu. "Improving Proximity Bounds Using Sparsity." In: *ISCO*. Vol. 12176. Lecture Notes in Computer Science. Springer, 2020, pp. 115–127.

[125] J. K. Lenstra, D. B. Shmoys, and É. Tardos. "Approximation algorithms for scheduling unrelated parallel machines." In: *Mathematical programming* 46.1 (1990), pp. 259–271.

[126] J. A. D. Loera, R. Hemmecke, S. Onn, and R. Weismantel. "n-fold integer programming." In: *Discrete Optimization* 5.2 (2008), pp. 231–241.

[127] A. Mallek, M. Bendraouche, and M. Boudhar. "Scheduling identical jobs on uniform machines with a conflict graph." In: *Computers & Operations Research* 111 (2019), pp. 357–366.

[128] K. L. Manders and L. M. Adleman. "NP-Complete Decision Problems for Binary Quadratics." In: *Journal of Computer and System Sciences* 16.2 (1978), pp. 168–184.

[129] S. Martello and P. Toth. *Knapsack problems*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Algorithms and Computer Implementations. John Wiley & Sons, Ltd., Chichester, 1990.

[130] D. Marx. "Parameterized Complexity and Approximation Algorithms." In: *The Computer Journal* 51.1 (2008), pp. 60–78.

[131] D. Marx and M. Pilipczuk. “Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask).” In: *STACS*. Vol. 25. LIPIcs. 2014, pp. 542–553.

[132] S. T. McCormick, S. R. Smallwood, and F. C. R. Spieksma. “A polynomial algorithm for multiprocessor scheduling with two job lengths.” In: *Mathematics of Operations Research* 26.1 (2001), pp. 31–49.

[133] M. Mnich and R. van Bevern. “Parameterized complexity of machine scheduling: 15 open problems.” In: *Computers & Operations Research* 100 (2018), pp. 254–261.

[134] M. Mnich and A. Wiese. “Scheduling and fixed-parameter tractability.” In: *Mathematical Programming* 154.1-2 (2015), pp. 533–562.

[135] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. “Matching is as easy as matrix inversion.” In: *Combinatorica* 7.1 (1987), pp. 105–113.

[136] M. Naor, L. J. Schulman, and A. Srinivasan. “Splitters and Near-Optimal Derandomization.” In: *FOCS*. 1995, pp. 182–191.

[137] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[138] R. Niedermeier and P. Rossmanith. “An efficient fixed-parameter algorithm for 3-Hitting Set.” In: *Journal of Discrete Algorithms* 1.1 (2003), pp. 89–102.

[139] C. H. Norton, S. A. Plotkin, and É. Tardos. “Using Separation Algorithms in Fixed Dimension.” In: *Journal of Algorithms* 13.1 (1992), pp. 79–98.

[140] S. Onn and P. Sarrabezolles. “Huge Unimodular n-Fold Programs.” In: *SIAM Journal on Discrete Mathematics* 29.4 (2015), pp. 2277–2283.

[141] J. Paat, R. Weismantel, and S. Weltge. “Distances between optimal solutions of mixed-integer programs.” In: *Mathematical Programming* 179.1 (2020), pp. 455–468.

[142] D. R. Page and R. Solis-Oba. “Makespan minimization on unrelated parallel machines with a few bags.” In: *Theoretical Computer Science* 821 (2020), pp. 34–44.

[143] C. H. Papadimitriou. “On the complexity of integer programming.” In: *Journal of the ACM* 28.4 (1981), pp. 765–768.

[144] J. Petersen. “Sur le théorèm de Tait.” In: *L’Intermédiare des Mathématiciens* 5.1 (1898), pp. 225–227.

[145] L. Rohwedder. “Algorithms for Integer Programming and Allocation.” PhD thesis. Kiel University, 2019.

[146] J. B. Rosser and L. Schoenfeld. “Approximate formulas for some functions of prime numbers.” In: *Illinois Journal of Mathematics* 6 (1962), pp. 64–94.

[147] P. Sanders, N. Sivadasan, and M. Skutella. “Online Scheduling with Bounded Migration.” In: *Mathematics of Operations Research* 34.2 (2009), pp. 481–498.

[148] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[149] M. Schroeder. “The Chinese Remainder Theorem and Simultaneous Congruences.” In: *Number Theory in Science and Communication*. Springer, 2009, pp. 235–243.

[150] R. Schultz, L. Stougie, and M. H. Van Der Vlerk. “Two-stage stochastic integer programming: a survey.” In: *Statistica Neerlandica* 50.3 (1996), pp. 404–416.

[151] P. Schuurman and G. J. Woeginger. “Polynomial time approximation algorithms for machine scheduling: Ten open problems.” In: *Journal of Scheduling* 2.5 (1999), pp. 203–213.

[152] H. Shachnai and T. Tamir. “On two class-constrained versions of the multiple knapsack problem.” In: *Algorithmica* 29.3 (2001), pp. 442–467.

[153] H. Shachnai and T. Tamir. “Polynomial time approximation schemes for class-constrained packing problems.” In: *Journal of Scheduling* 4.6 (2001), pp. 313–338.

[154] P. W. Shor. “Random planar matching and bin packing.” PhD thesis. Massachusetts Institute of Technology, Department of Mathematics, 1985.

[155] P. W. Shor. “The average-case analysis of some on-line algorithms for bin packing.” In: *Combinatorica* 6.2 (1986), pp. 179–200.

[156] M. Skutella and J. Verschae. “Robust Polynomial-Time Approximation Schemes for Parallel Machine Scheduling with Job Arrivals and Departures.” In: *Mathematics of Operations Research* 41.3 (2016), pp. 991–1021.

[157] M. Sorge, R. van Bevern, R. Niedermeier, and M. Weller. “A new view on rural postman based on Eulerian extension and matching.” In: *Journal of Discrete Algorithms* 16 (2012), pp. 12–33.

[158] K. Subramani. “On deciding the non-emptiness of 2SAT polytopes with respect to First Order Queries.” In: *Mathematical Logic Quarterly* 50.3 (2004), pp. 281–292.

[159] É. Tardos. “A Strongly Polynomial Algorithm to Solve Combinatorial Linear Programs.” In: *Operations Research* 34.2 (1986), pp. 250–256.

[160] M. Tutas. *Approximation Algorithms and Inapproximability of Scheduling Problems with Different Variants of Class Constraints*. Bachelor Thesis. Supervised by Klaus Jansen and Alexandra Lassota. 2020.

[161] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[162] D. E. D. Vinkemeier and S. Hougardy. "A linear-time approximation algorithm for weighted matchings in graphs." In: *ACM Transactions on Algorithms* 1.1 (2005), pp. 107–122.

[163] S. Wagon. *Mathematica in action*. Springer Science & Business Media, 1999.

[164] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[165] G. J. Woeginger. "There is no asymptotic PTAS for two-dimensional vector packing." In: *Information Processing Letters* 64.6 (1997), pp. 293–297.

[166] E. C. Xavier and F. K. Miyazawa. "The class constrained bin packing problem with applications to video-on-demand." In: *Theoretical Computer Science* 393.1-3 (2008), pp. 240–259.

ERKLÄRUNG ÜBER DEN EIGENANTEIL AN PUBLIKATIONEN MIT MEHREREN AUTOREN

Die Ideen und Beweise der Arbeit [96] hat Frau Alexandra Lassota zu mehr als der Hälfte ausgearbeitet und die Arbeit fast vollständig selbstständig und alleine geschrieben. Bei der Erstellung der Manuskripte hat Sie Ihr Betreuer Prof. Jansen entsprechend unterstützt.

Bei den beiden Arbeiten [100, 19] hat Frau Alexandra Lassota die Hälfte der Ideen für die Algorithmen und Beweise entwickelt und mehr als die Hälfte der Arbeit aufgeschrieben. Bei der Erstellung der Manuskripte hat Sie Ihr Betreuer Prof. Jansen entsprechend unterstützt.

Bei der Arbeit [102] hat Frau Alexandra Lassota zu gleichen Teilen mit Herrn Lars Rohwedder die Ideen für die Algorithmen und die strukturellen Aussagen entwickelt und die Beweise und Details ausgearbeitet. Auch der Aufschrieb wurde gleichmäßig aufgeteilt. Bei der Erstellung des Manuskripts hat Sie Ihr Betreuer Prof. Jansen entsprechend unterstützt.

Die algorithmischen Ideen und Beweisskizzen der Arbeit [14] sind während eines Workshops in Bergen (Operations Research + Parameterized Complexity Workshop, 2018) mit allen Koautoren in enger Zusammenarbeit entstanden. Beim Ausarbeiten der Details und der Beweise sowie dem Aufschrieb hat Frau Lassota mehr als die Häfte zu den Kapiteln 2 und 3 beigetragen und auch beim Überarbeiten des ganzes Werkes maßgeblich mitgewirkt.

Bei der Arbeit [101] hat Frau Alexandra Lassota die Hälfte zu den FPT Ergebnissen (Kapitel 4) des Manuskriptes beigetragen. Bei dem Aufschrieb dieser Teile hat sie mehr als die Hälfte beigetragen. Die anderen Kapitel basieren vorrangig auf den Ideen von Herrn Tytus Pikies und Herrn Marten Maack und wurden deswegen nicht in die Doktorarbeit mit aufgenommen.

Die Ideen und Beweisskizzen des Manuskriptes [55] sind während einer Reise von Frau Alexandra Lassota und Herrn Lars Rohwedder nach Haifa (April/ Mai 2019) und während des Gegenbesuchs von Frau Leah Epstein und Herrn Asaf Levin an der Christian-Albrechts-Universität zu Kiel (August/ September 2019) in intensiver Zusammenarbeit mit allen Koautoren entstanden. Die Teile, die in diese Dissertation aufgenommen wurden, wurden selbstständig von Frau Alexandra Lassota ausgearbeitet und niedergeschrieben.

ERKLÄRUNG

Hiermit gebe ich folgende Erklärungen ab:

- Diese Abhandlung ist, abgesehen von der Beratung durch meinen Betreuer Klaus Jansen, nach Inhalt und Form meine eigene Arbeit. Ich habe sie eigenständig und nur mit den angegebenen Hilfsmitteln verfasst.
- Die Arbeit ist unter Einhaltung der Regeln guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft entstanden.
- Es wurde mir noch nie ein akademischer Grad entzogen.

Teile dieser Arbeit sind bereits an anderer Stelle im Rahmen eines Prüfungsverfahrens vorgelegt worden. Dies betrifft Kapitel 3, welches auf der Arbeit [102] basiert und in ähnlicher Form auch in der Dissertation meines Mitautors Lars Rohwedder [145] enthalten ist.

Des Weiteren kann dies Kapitel 7 betreffen, welches auf der Arbeit [101] basiert, da mein Mitautor Tytus Pikies plant Teile dieser gemeinsamen Arbeit mit in seine Dissertation aufzunehmen und zeitnah einzureichen.

Kein anderer Teil dieser Arbeit ist bereits an anderer Stelle im Rahmen eines Prüfungsverfahrens vorgelegt worden. Teile wurden, wie in der Arbeit gekennzeichnet, im Rahmen wissenschaftlicher Veröffentlichungen publiziert.

Kiel, März 2021

Alexandra Anna Lassota

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić with some modifications by the author. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Version as of July 5, 2021 (classicthesis v4.6).