About the Structure and Sensitivity of Integer Linear Programs and their Application in Combinatorial Optimization

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

Dipl.-Inf. Kim-Manuel Klein

Kiel
2017

1. Gutachter: Prof. Dr. Klaus Jansen, Christian-Albrechts-Universität zu Kiel
2. Gutachter: Prof. Dr. Martin Skutella, Technische Universität Berlin
3. Gutachter: Prof. Dr. Monaldo Mastrolilli, IDSIA Lugano

Datum der Disputation: 19. April 2017
Zum Druck genehmigt: 19. April 2017

Zusammenfassung

In dieser Dissertation werden Eigenschaften von ganzzahligen linearen Programmen (engl. integer linear programs, kurz: ILPs) untersucht. Von Interesse sind dabei hauptsächlich ILP-Formulierungen, welche sich aus dem Kontext von algorithmischen Problemstellungen ergeben, wie beispielsweise dem Bin Packing-Problem und dem Scheduling-Problem auf identischen Maschinen. Insbesondere für diese ILPs zeigen wir Strukturaussagen, sowie Aussagen über die Sensitivität und können so offene algorithmische Fragestellungen im Bereich von Approximation und parametrisierter Komplexität lösen.

Im Kontext von Sensitivitätsaussagen wird untersucht, inwiefern Lösung des ILPs angepasst werden können, wenn sich die Parameter des ILPs leicht ändern. Ein klassisches Resultat von Cook u.a. [Coo+86] gibt dabei für *optimale* Lösungen des ILPs Abschätzungen an. In dieser Arbeit betrachten wir Abschätzungen für die Senstivität wenn *approximative* Lösungen erlaubt sind, d.h. Lösungen deren Zielfunktionswert höchstens um einen Faktor $1+\epsilon$ über dem optimalen Zielfunktionswert liegt. Im Wesentlichen zeigen wir, dass sich bei approximative Lösungen die Abschätzungen von Cook u.a. verbessern lassen. Diese Ergebnisse konnten wir auf das Online-Bin Packing-Problem anwenden, wenn eine approximative Lösung mit Güte $1+\epsilon$ erreicht werden soll und in beschränktem Maße Items umgepackt werden dürfen. Durch unsere neuen Sensitvitätsabschätzungen konnten wir so existierende Ergebnisse verbessern, welche einen Algorithmus mit exponentiellem Term in $1/\epsilon$ beim Umpacken benötigen. Unser Algorithmus ist tatsächlich der erste Algorithmus für ein NP-vollständiges Problem, der die exponentielle Migration vermeiden kann, welcher aus der Anwendung des Theorems von Cook u.a. resultiert, und lediglich polynomielle Migration benötigt. Desweiteren stellen wir einen Algorithmus für das sogenannte vollständig dynamische Bin Packing-Problem vor. Im Gegensatz zum klassischen Online Bin Packing-Problem können hierbei Lösch-Operationen auftreten, bei denen Items aus der vorhanden Packung entfernt werden müssen.

Im Kontext von Strukturaussagen wird in dieser Dissertation die Existenz von Lösungen einer Klasse von ILPs gezeigt (sofern überhaupt eine Lösung existiert), welche eine bestimmte vereinfachte Struktur aufweisen. Diese Existenzaussagen sind sehr hilfreich um eine Lösung des ILPs zu finden, da der Suchraum auf Lösungen dieser bestimmten Struktur eingeschränkt werden kann. Wir konnten Strukturaussagen für ILPs entwickeln, welche sich aus Formulierungen des Bin Packing-Problems ergeben bzw. natürliche Verallgemeinerungen dieser Formulierung. Dadurch ist es uns zum einen gelungen ein effizientes Approximationsschemata für das Scheduling-Problem auf identischen Maschinen mit einer Laufzeit von $2^{\tilde{O}(1/\epsilon)} + \text{poly}(n)$ zu entwicklen. Damit konnten wir die Laufzeit von bisherigen Algorithmen verbessern und die Lücke zur unteren Laufzeitschranke von Chen u.a. [CJZ13] basierend auf der sogenannte Exponentialzeit-Hypothese bis auf logarithmische Terme zu schließen. Desweiteren konnten wir eine Strukturaussage entwickeln, welche unter anderem Anwendung im Bin Packung-Problem fand, wenn die Anzahl der unterschiedlichen Itemgrößen d beschränkt ist. Wir konnten zeigen, dass stets eine optimale Lösung existiert, welche fast ausschließlich (bis auf konstant viele) die Ecken des zugrundeliegenden ganzzahligen Rucksack-Polytops V_I verwendet. Mit Hilfe dieser Strukturaussage konnten wir einen fpt-Algorithmus für das Bin Packing-Problem entwickeln mit einer Laufzeit von $|V|^{2^d} enc(I)$, wobei V die Menge der Ecken des ganzzahligen Rucksack-Polytops ist und $enc(I)$ die Kodierungslänge der Eingabeinstanz.

Abstract

In this thesis we investigate properties of integer linear programs (ILPs) and their algorithmic use. Our main focus are ILP-formulations that come from concrete algorthmic problems like the bin packing problem or the scheduling problem on identical machines. Especially for this kind of ILPs we study structural properties as well as properties for their sensitivity. As a result, we are able to answer open algorithmical questions in the area of approximation and parameterized complexity.

In the context of sensitivity we analyze how much an ILP solution has to be adjusted when the parameters of the ILP change. There is a classical results by Cook et al. [Coo+86] which gave bounds for that question when optimal solutions are considered. However, in this thesis we investigate the sensitivity of ILPs when approximate solutions are allowed, i.e. solutions that differ by a factor of at most $(1+\epsilon)$ from the optimum value. In this case of approximate solutions, we were able to show that the bounds derived from the theorem by Cook et al. can be improved in some sense. And furthermore, we could apply the obtained results to the online bin packing problem, when an approximation guarantee with ratio $1+\epsilon$ has to be fulfilled and repacking of already assigned items (limited by the so called migration factor) is allowed. Using our sensitivity results, we were able to improve upon existing results, which had an exponential term in $1/\epsilon$ in the migration of items. In fact, our result was the first result for an NP-hard problem that could avoid the exponential term implied by the use of Cook et al. and obtain only polynomial migration. As a further result, we present an algorithm for the so called fully dynamic bin packing. This model is a generalization of the classical online setting in which there are also delete operations where items have to be departed from the packing.

In the context of structural results, we prove the existence (assuming the ILP is feasible) of solutions of a certain class of ILPs with a certain simplified structure. For example, we prove the existence of solutions that use a specific type of variables very often. The knowledge about the existence of solutions with this simplified structure can be very useful for solving the ILP as the search space for the set of possible solutions is reduced. In this thesis we prove structure properties for ILPs that arise from formulations of bin packing or scheduling problems and natural generalization of those formulations. Based on the those structure properties, we develop an efficient approximation scheme for the scheduling problem on identical machines with a running time of $2^{\tilde{O}(1/\epsilon)} + \text{poly}(n)$. With this algorithm we can improve upon previous results and basically close the gap to the lower bound in the running time, which is based on the exponential time hypothesis (ETH). Furthermore, we develop a structure theorem, which is applied to the bin packing problem when the number of different item sizes d is bounded. We basically prove that there is always an optimal solution that uses (except for a constant number of components) only the vertices of the underlying integral knapsack polytope. Applying this structure result, yields an fpt algorithm for the bin packing problem with running time $|V|^{2^d} enc(I)$, where V is the set of all edges of the integral knapsack polytope and $enc(I)$ is the encoding length of the instance.

Acknowledgments

First, I would like to thank my advisor Klaus Jansen for his patience with me, his support and insights. Thanks also to my coauthor José Verschae who invited me to Chile where I really had a great time doing research together with him. And I would like to thank my coauthor and friend Sebastian Berndt for a good and fun collaboration.

I am greatful to all my colleagues at the university Stefan Kraft, Marten Maack, Ute Iaquinto, Maren Kaluza, Kati and Felix Land, Parvaneh Karimi Massouleh, Marcin Pal, Lars Prädel, Malin Rau, Christina Robenek, and Ilka Schnoor for all the motivating discussions and the good times we had together.

Finally, of course I thank my wife Gina Klein, my family and all my friends here in Kiel and in Stuttgart.

Contents

Publications

The results of this thesis have appeared in the following publications:

[BJK15] S. Berndt, K. Jansen, and K. Klein. "Fully Dynamic Bin Packing Revisited". In: *18th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. 2015, pp. 135–151.

[JK] K. Jansen and K. Klein. "About the Structure of the Integer Cone and its Application to Bin Packing". In: *Symposium on Discrete Algorithms, SODA (to appear)*.

[JK13] K. Jansen and K. Klein. "A Robust AFPTAS for Online Bin Packing with Polynomial Migration". In: *International Colloquium on Automata, Languages, and Programming(ICALP)*. 2013, pp. 589–600.

[JKV16] K. Jansen, K. Klein, and J. Verschae. "Closing the Gap for Makespan Scheduling via Sparsification Techniques". In: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*. 2016, 72:1–72:13.

1 Introduction and Outline of the Thesis

In the following we give an introduction and an outline of the thesis. Still, each of the following chapters is self sufficient and contains a separate introduction where the respective results are motivated and related work is mentioned.

Consider an integer linear program (ILP) of the form

$$\min\{\|x\|_1 \mid Ax \geq b, x \in \mathbb{Z}_{\geq 0}\},$$

for some matrix $A \in \mathbb{Z}^{m \times d}$ and some vector $b \in \mathbb{Z}^d$. Integer programming is a fundamental tool in the design of algorithms. Plenty of problems can be formulated by an ILP in a way that a solution of the formulated ILP gives a solution of the problem. Especially in combinatorial optimization, the use of ILPs has become very common in the design of algorithms. In this algorithmic context, several natural question appear regarding properties of ILPs. The first question would certainly be on how efficient a solution of the given ILP can be computed. For general ILPs, Kannan [Kan87] provides an algorithm with a good worst case complexity compared to the best known lower bound. However, when it comes to ILP formulations that arise from a concrete algorithmic problem, one might question if a direct use of the algorithm by Kannan is really the most efficient way to solve the given ILP. One approach that we follow in this thesis, is to study so called structural properties of a class of ILP formulations. By proving that there is a solution of the given ILP with a certain structure, we are able to show that this class of ILPs can be computed more efficiently. Another property of ILPs that we consider in this thesis arises in the context of online algorithms. In online algorithms, the problem instance changes over time, as for example new information is available. And as the problem instance changes over time, so does the corresponding ILP formulation of the problem. This brings up the question of the sensitivity of an ILP, which states if the current solution of an ILP can be adjusted (with as small changes as possible) to a modified ILP, with slightly changed parameters.

Applying our results from integer programming, we study two of the most famous and fundamental problems from combinatorial optimization. One problem is the so called *bin packing* problem. In the bin packing problem a set of n items I is given, where each item $i \in I$ has a size $s(i) \in (0, 1]$. The objective of the problem is to find a packing of the items into as few unit sized bins as possible. The other problem that we consider is the so called *scheduling* problem on identical machines. In this problem a set of n jobs $\mathcal{J}$ and a set of m identical machines $\mathcal{M}$ is given, where each job $j \in \mathcal{J}$ has processing time $p_j \in \mathbb{Z}_{>0}$. The load of a machine is the total processing time of jobs assigned to it and our objective is to minimize the *makespan*, that is, the maximum machine load over all machines.

Problems in combinatorial optimization are often NP-hard and so are the bin packing problem and the scheduling problem. This means that under the assumption that $P \neq NP$, there is no polynomial time algorithm to solve these problems to optimality. Therefore, those problems are often considered in the context of approximation, where solutions

are allowed that are only close to the optimum solution. We say an algorithm (for a minimization problem) has an *absolute* approximation guarantee of α, if for every input I of the problem, the algorithm returns a solution with $A(I) \leq \alpha \cdot OPT(I)$, where $OPT(I)$ is the value of the optimum solution and $A(I)$ is the value of the solution that is returned by the algorithm. We say that an algorithm has an *asymptotic* approximation guarantee of α if $\alpha = \limsup_{x\to\infty} \sup\{\frac{A(I)}{OPT(I)} \mid OPT(I) = x\}$. This leads us to the notion of *(asymptotic) polynomial time approximation schemes ((A)PTAS)*. A problem admits a (A)PTAS if for every $\epsilon \in (0, 1]$ there exists a polynomial time algorithm A_ϵ such that A_ϵ has an (asymptotic) approximation guarantee of $1 + \epsilon$.

In Section 2–4, we study the scheduling problem and online bin packing with migration in the context of approximation. Another approach to tackle NP-hard problems that we use in chapter 5 is *parameterized complexity*. In parameterized complexity problems are classified according to their difficulty with respect to certain parameters of the input instance. Therefore, we say an algorithm is fixed parameter tractable (fpt) for a parameter p if there exists an algorithm with running time $f(p) \cdot enc(I)^{O(1)}$ for some computable function f of p which is independent of I and $enc(I)$.

1.1 Sensitivity and its Application to Online Bin Packing

Let ILP (1) be of the form $\min\{\|x\|_1 \mid Ax \geq b, x \in \mathbb{Z}_{\geq 0}\}$ with an optimal solution y and let ILP (2) be an ILP with changed right hand side b'. The question of the sensitivity is whether there exists a solution y' of ILP (2) such that the distance $\|y' - y\|_\infty$ between the old solution y and the new solution y' is possibly small. This question was already studied by Cook et al. in 1986. The theorem by Cook et al. [Coo+86] states that for the case that ILP (2) is feasible, there exists a solution y' with $\|y' - y\|_\infty \leq n\Delta(\|b' - b\|_\infty + 2)$, where Δ is the largest subdeterminant of the matrix A. Sanders et al. [SSS09] found out that this classical theorem of Cook et al. can be applied in the setting of online algorithms. They presented an online algorithm for the scheduling problem on identical machines when reassigning already assigned jobs is allowed. To give a measure on how many items are allowed to be reassigned, Sanders et al. [SSS09] defined the *migration factor*. It is defined by the complete size of all moved jobs divided by the size of the arriving one. As a result, Sanders et al. obtained a PTAS with a migration factor of $2^{\mathcal{O}(\frac{1}{\epsilon}\log^2\frac{1}{\epsilon})}$.

1.1.1 A Robust AFPTAS for Bin Packing

Chapter 2 of this thesis is based on [JK13]. In this respective chapter we show that an exponential term in the migration can be avoided in case of the bin packing problem. As the use of the theorem of Cook et al. [Coo+86] leads to an exponential term in the migration, we develop new ILP techniques that are based on the relaxed linear program and approximate solutions. In the following we state our central observation which basically states that we can find a feasible LP solution x'' with improved objective value such that $\|x'' - x'\|_1$ is rather small.

Theorem. *Consider the linear program* $\min\{\|x'\|_1 | Ax \geq b, x \geq 0\}$ *and an approximate solution* x' *with* $\|x'\|_1 \leq (1 + \delta)LIN$ *for some* $\delta > 0$*, where LIN is the optimum value of the LP. For every positive* $\alpha \leq \delta LIN$ *there exists a solution* x'' *with objective value of at most* $\|x''\|_1 \leq (1 + \delta)LIN - \alpha$ *and distance* $\|x' - x''\|_1 \leq 2\alpha(1/\delta + 1)$.

It is easy to see that this theorem is useful in the analysis of sensitivity as the solution with the improved objective value can be modified to cover a bigger right hand side. In chapter 2, we build upon this theorem to find similar observations for ILPs and give an perspective on how these observations can be applied algorithmically. As a result, we present an APTAS for the bin packing problem with a migration factor of $O(\frac{1}{\epsilon^4})$, which improves upon the algorithm by Epstein and Levin [EL09] who gave an APTAS for the bin packing problem with a migration factor of $2^{\mathcal{O}(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})}$.

1.1.2 Fully Dynamic Bin Packing

In chapter 3, which is based on [BJK15], we show that we can generalize the results in chapter 2 to an online bin packing model, where also depart operations are allowed, such that items have to be removed from the current packing. This model is called fully dynamic bin packing.

In this chapter we essentially present new dynamic rounding techniques such that the ILP techniques of chapter 2 can be applied. Furthermore, we discuss the difficulty of small items in the fully dynamic setting. As a result we develop an asymptotic PTAS for the problem with a migration factor of $O(\frac{1}{\epsilon^4}\log\frac{1}{\epsilon^4})$ and therefore improve upon the result by Ivković and Lloyd [IL98] who gave a $5/4$ approximation (rather than $1+\epsilon$) for the problem using $O(\log n)$ so called shifting moves. Furthermore, we give a lower bound on the migration that is needed for an APTAS.

1.2 Structural Results and their Application

A major open problem in the field of combinatorial optimization was if the bin packing problem can be solved in polynomial time when the number of different item sizes is constant. Very recently Goemans and Rothvoss [GR14] gave a positive answer to that question. They presented a polynomial time algorithm for the problem which relies on very novel ideas for structure analysis and how this structure can be exploited to obtain polynomial time algorithms. Moreover, their techniques and algorithms do not only apply for the bin packing problem. They can be applied to a wide range of combinatorial problems like preemptive scheduling or scheduling on heterogeneous machines.

In chapter 4 and 5 of this thesis we build upon the techniques by Goemans and Rothvoss and present new structural results for the following ILP, which is defined for a given Polytope $\mathcal{P} \subset \mathbb{R}^d$:

$$\begin{aligned} &\min \|x\|_1 \\ \sum_{p\in\mathcal{P}\cap\mathbb{Z}^d} &x_p p = b \\ &x_p \in \mathbb{Z}_{\geq 0} \end{aligned} \tag{1.1}$$

Note that in the case that $\mathcal{P}$ is the knapsack polytope (i.e. $\mathcal{P} = \{x \in \mathbb{R}^d \mid s_1x_1 + \ldots + s_dx_d \leq 1\}$ for sizes $s_1, \ldots s_d \in (0,1]$) , this ILP solves the bin packing problem. We show that the presented structure results can be applied to the scheduling problem on identical machines (as well as some generalizations of that model) and the bin packing problem to obtain new efficient algorithms.

1.2.1 Scheduling on Identical Machines

As a main result of chapter 4, which is based on [JKV16], we obtain the following structure result for ILP 1.1, when $\mathcal{P} \subset \mathbb{R}^d$ is the knapsack polytope. Let $supp(x)$ be the number of non-zero components of a given solution x of ILP (1.1).

Theorem. *If there exists a solution of ILP (1.1), where $\mathcal{P}$ is the knapsack polytope and T is the maximum entry of vectors in $\mathcal{P} \cap \mathbb{Z}^d$,then there exists also a solution x such that*

1. *if $x_c > 1$ then $c \in \mathcal{P}$ contains $\leq \log(T+1)$ different item sizes,*
2. *$|supp(x)| \leq 4(d+1)\log(4(d+1)T)$,*
3. *$\sum_{c \in Q_c} x_c \leq 2(d+1)\log(4(d+1)T)$, where $Q_c \subset \mathcal{P}$ denotes the set of configurations c with more than $\log(T+1)$ different item sizes.*

Using this theorem, we present a PTAS for the scheduling problem on identical machines with a running time $2^{O((1/\epsilon)\log^4(1/\epsilon))} + O(n \log n)$. This improves upon the previous best known PTAS by Jansen [Jan10] and basically closes the gap to the lower bound by Chen et al. [CJZ13] which states that there is no PTAS with a running time of $2^{(1/\epsilon)^{1-\delta}} + \text{poly}(n)$ for every $\delta > 0$ assuming the exponential time hypothesis [CJZ13].

1.2.2 Bin Packing in fpt time

Based on [JK], we revisit in chapter 5 the structure theorem by Goemans and Rothvoss [GR14] and prove the following structure result:

Theorem. *Let V_I be the vertices of the intergral knapsack polytope. If there exists a solution of ILP (1.1), then there exists also a solution $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$ such that*

1. *$\lambda_p \leq 2^{2^{O(d)}} \quad \forall p \in (\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$*
2. *$|supp(\lambda) \cap V_I| \leq d \cdot 2^d$*
3. *$|supp(\lambda) \setminus V_I| \leq 2^{2d}$*

As a consequence of this structure theorem we obtain an algorithm for the bin packing problem with running time $|V|^{2^{O(d)}} \cdot enc(I)^{O(1)}$, where V is the set of vertices of the integer knapsack polytope and $enc(I)$ is the encoding length of the bin packing instance. The algorithm is fixed parameter tractable, parameterized by the number of vertices of the integer knapsack polytope $|V|$. This shows that the bin packing problem can be solved efficiently when the underlying integer knapsack polytope has an easy structure, i.e. has a small number of vertices. This is for example the case when all item sizes divide 1 (i.e. $1/2, 1/3, 1/4, \ldots$). However, in any case, the total number of vertices is bounded by $O(\log \Delta)^d$ [HL83]. Therefore our algorithm has a worst case running time of $(\log \Delta)^{2^{O(d)}}$, which is identical to the running time of the algorithm by Goemans and Rothvoß [GR14]. In chapter 5 we show furthermore that the presented bounds of the structure theorem are asymptotically tight. We give a construction of bin packing instances using new structural insights and classical number theoretical theorems which yield the desired lower bound.

2 A Robust AFPTAS for Online Bin Packing with Polynomial Migration

2.1 Introduction

The idea behind robust algorithms is to find solutions of an optimization problem that are not only good for a single instance, but also if the instance changes in certain ways. Instances change for example due to uncertainty or when new data arrive. With changing parameters and data, we have the effort to keep as much parts of the existing solution as possible, since modifying a solution is often connected with costs or may even be impossible in practice. Achieving robustness especially for linear programming (LP) and integer linear programming (ILP) is thus a big concern and a very interesting research area. Looking at worst case scenarios, how much do we have to modify a solution if the LP/ILP is changing? There is a result of Cook et al. [Coo+86] giving an upper bound for ILPs when changing the right hand side of the ILP. Many algorithms in the theory of robustness are based on this theorem.

As a concrete application we consider the classical online bin packing problem, where it is additionally allowed to reassign already packed items. In this setting, items of size $s_i \in (0, 1]$ arrive over time and our objective is to assign these items into as few unit sized bins as possible. As soon as a new item arrives, it is allowed to replace a set of already packed items with bounded total size.

In the case of offline bin packing it is known that unless $\mathcal{P} = \mathcal{NP}$ there is no polynomial time approximation algorithm for offline bin packing that produces a solution better than $\frac{3}{2}OPT$, where OPT is the minimum number of bins needed. The most common way to measure the approximation guarantee of an algorithm for the bin packing packing problem is the *asymptotic approximation ratio*. The asymptotic approximation ratio for an algorithm A is defined by $\limsup_{x\to\infty} \sup\{\frac{A(I)}{OPT(I)} \mid OPT(I) = x\}$. This leads to the notion of *asymptotic polynomial time approximation schemes (APTAS)*. Given an instance of size n and a fixed parameter $\epsilon \in (0, 1]$, an APTAS has a running time of $poly(n)$ and asymptotic approximation ratio $1+\epsilon$. A typical running time for this class of algorithms is $\mathcal{O}(n^{f(\frac{1}{\epsilon})})$ for an arbitrary function f. An APTAS is called an *asymptotic fully polynomial time approximation scheme (AFPTAS)* if its running time is polynomial in n *and* in $\frac{1}{\epsilon}$. The first APTAS for offline bin packing was developed by Fernandez de la Vega & Lueker [FL81], and Karmakar & Karp improved this result by giving an AFPTAS [KK82] (see survey on bin packing [CGJ97]).

Since the introduction by Ullman of the classical online bin packing problem [Ull71], there has been plenty of research (see survey [CW96] or [Cof+13] for an extensive survey on various models). The best known algorithm has an asymptotic competitive ratio of 1.5815 [HS16] compared to the optimum in the offline case, while the best known lower bound is 1.54037 [BBG12]. Due to the relatively high lower bound of the classical online bin packing problem, there has been effort to extend the model with the purpose to obtain an improved competitive ratio. Gambosi et al. [GPT00] presented a model where

they allow repacking of items. They presented an algorithm which achieves ratio 1.33 by using at most 7 shifting moves each time a new item arrives. A shifting move consists of moving a large item or a bundle of small items from one bin to another. Ivkovic and Lloyd [IL98] presented an algorithm for fully dynamic bin packing having ratio 1.25. In this online model, items may arrive and also depart. Their algorithm requires $\mathcal{O}(\log n)$ shifting moves. We revisit the fully dynamic model in chapter 3 of this thesis. In another work Ivkovic and Lloyd [IL97] gave an algorithm which achieves approximation ratio $1+\epsilon$ by using amortized $\mathcal{O}(\log n)$ shifting moves. On the other hand, Balogh et al. [Bal+08] proved that a lower bound of 1.3871 on the competitive ratio holds if an algorithm moves at most $\mathcal{O}(1)$ items. Note that this lower bound does not contradict to the mentioned results, since a shifting move may contain $\Theta(n)$ very small items.

The model we follow is the notion of the migration factor. Introduced by Sanders et al. [SSS09] it allows repacking of arbitrary items while the number of items that are being repacked is limited. To give a measure on how many items are allowed to be repacked Sanders et al. [SSS09] defined the *migration factor*. It is defined by the complete size of all moved items divided by the size of the arriving one. An (A)PTAS is called *robust* if its migration factor is of size $f(\frac{1}{\epsilon})$, where f is an arbitrary function that only depends on $\frac{1}{\epsilon}$. Since the promising introduction of robustness, several robust algorithms have been developed. Sanders et al. [SSS09] found a robust PTAS for the online scheduling problem on identical machines, where the goal is to minimize the makespan. The robust PTAS has constant but exponential migration factor $2^{\mathcal{O}(\frac{1}{\epsilon}\log^2\frac{1}{\epsilon})}$. In case of bin packing Epstein and Levin [EL09] developed a robust APTAS for the classical bin packing problem with migration factor $2^{\mathcal{O}(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})}$ and running time double exponential in $\frac{1}{\epsilon}$. In addition they proved that there is no optimal online algorithm with a constant migration factor. Furthermore, Epstein and Levin [EL13] showed that the robust APTAS for bin packing can be generalized to packing d-dimensional cubes into a minimum number of unit cubes. Recently Epstein and Levin [EL14] also designed a robust algorithm for preemptive online scheduling of jobs on identical machines, where the corresponding offline problem is polynomial solvable. They presented an algorithm with migration factor $1-\frac{1}{m}$ that computes an optimal solution whenever a new item arrives. Skutella and Verschae [SV16] studied the problem of maximizing the minimum load given n jobs and m machines. They proved that there is no robust PTAS for this machine covering problem with constant migration. On the positive side, they gave a robust PTAS for the machine covering problem in the case that migrations can be reserved for a later timestep. The algorithm has an amortized migration factor of $2^{\mathcal{O}(\frac{1}{\epsilon}\log^2\frac{1}{\epsilon})}$.

2.1.1 Our Results:

An online algorithm is called *fully robust* if its migration factor is bounded by $p(\frac{1}{\epsilon})$, where p is a polynomial in $\frac{1}{\epsilon}$. The purpose of this chapter is to give methods to develop fully robust algorithms. In Section 2 we develop a theorem for a given linear program (LP) $\min\{\|x\|_1 \mid Ax \geq b, x \geq 0\}$. Given an approximate solution x' with value $(1+\delta)LIN$ (where LIN is the minimum objective value of the LP) and a parameter $\alpha \in (0, \delta LIN]$, we prove the existence of an improved solution x'' with value $(1+\delta)LIN - \alpha$ and distance $\|x''-x'\|_1 \leq \alpha(2/\delta+2)$. Furthermore, we extend these techniques to obtain similar results for integral solutions of the ILP. Therefore, we define a correspondence between a given fractional solution x' and an integral solution y'. Then we are able to show the existence of an improved integral solution y'' with $\|y''-y'\|_1 = \mathcal{O}(\frac{\alpha+m}{\delta})$ (where m is the

number of inequalities of the ILP). Since both results are constructive, we propose also algorithms to compute such improved solutions. Previous robust online algorithms require an optimum solution of the corresponding ILP and use a sensitivity theorem by Cook et al. [Coo+86]. This results in an exponential migration factor in $\frac{1}{\epsilon}$ ([EL09; EL13; SV10; SSS09]). In contrast to this we consider approximate solutions of the corresponding LP relaxations and are able to use the techniques above to improve the fractional and integral solutions. Furthermore we also prove an approximate version of a sensitivity theorem for LPs with modified right hand side b and b'. During the online algorithm the number of non-zero variables increases from step to step and would result in a large additive term. To avoid this, we present algorithms in Section 2.3 to control the number of non-zero variables of the LP and ILP solutions. We can bound the number of non-zero variables and the additive term by $\mathcal{O}(\epsilon LIN) + \mathcal{O}(\frac{1}{\epsilon^2})$.

In Section 2.4, we use the developed ILP techniques to develop a fully robust AFPTAS for the robust bin packing problem. There is a natural ILP formulation Eisemann [Eis57] for the problem such that each item size corresponds to one constraint of the ILP. Hence, to actually obtain an ILP formulation with bounded size and use the developed techniques from Section 2.2 and Section 2.3, we need to use a dynamic rounding. In that, the sizes of the items are rounded such that the total number of different item sizes is bounded over all time steps of the online algorithm. Therefore, as soon as a new item arrives, the dynamic rounding decides to which value the new item is rounded. Our presented dynamic rounding is a modification of the rounding techniques by Epstein and Levin [EL09]. However, one difficulty that we have to overcome is that we use approximate solutions of the LP with corresponding integral solutions rather than only optimal solutions of the ILP. This leads to a rather sophisticated analysis of the algorithm. Finally, by combining the dynamic rounding and the algorithm to get improved solutions of the LP and ILP, we are able to obtain a fully robust AFPTAS for the online bin packing problem. The algorithm has a migration factor of $\mathcal{O}(1/\epsilon^4)$ (or $\mathcal{O}(1/\epsilon^3)$ if the size of the arriving item is $\Omega(1)$) and running time polynomial in $\frac{1}{\epsilon}$ and t, where t is the number of arrived items. This resolves an open question of Epstein and Levin [EL09]. We believe that our techniques can be used for other online problems to obtain algorithms with low migration factors.

2.2 Robustness of approximate LPs

We consider a matrix $A \in \mathbb{R}^{m \times n}_{\geq 0}$, a vector $b \in \mathbb{R}^m_{\geq 0}$ and a cost vector $c \in \mathbb{R}^n_{\geq 0}$. The goal in a *linear program* (LP) is to find a $x \geq 0$ with $Ax \geq b$ such that the *objective value* $c^T x$ is minimal. We say x^{OPT} is an optimal solution if $c^T x^{OPT} = \min\left\{c^T x | Ax \geq b, x \geq 0\right\}$ and we define $LIN = c^T x^{OPT}$. In general we suppose that the objective function of a solution is positive and hence $LIN > 0$. We say x' is an approximate solution with approximation ratio $1 + \delta$ for some $\delta \in (0, 1]$ if $c^T x' \leq (1 + \delta) LIN$. We will present Lemma 2.1 with a general objective value c, but for the remaining part of the chapter we will assume that $c^T = (1, 1, \ldots, 1)$ and therefore $c^T x^{OPT} = \left\|x^{OPT}\right\|_1 = LIN$. The following theorem is central. Given an approximate solution x', we want to find a solution x'' with improved objective value. To achieve robustness we have to provide a small distance between x' and x''. In the case of $c^T = (1, 1, \ldots, 1)$ we prove that there is an approximate solution x'' with improved objective value $\leq (1 + \delta) LIN - \alpha$ within distance $\|x' - x''\|_1 = \mathcal{O}(\frac{\alpha}{\delta})$.

Lemma 2.1. *Consider the LP* $\min\left\{c^T x | Ax \geq b, x \geq 0\right\}$ *and an approximate solution*

x' with $c^T x' = (1+\delta)LIN$ for some $\delta > 0$. For every positive $\alpha \leq \delta LIN$ there exists a solution x'' with objective value of at most $c^T x'' \leq (1+\delta)LIN - \alpha$ and distance $\|x' - x''\|_1 \leq \alpha(1/\delta+1)\frac{\|x'\|_1 + \left\|x^{OPT}\right\|_1}{c^T x'}$. If $c^T = (1,1,\ldots,1)$ then $\|x' - x''\|_1 \leq 2\alpha(1/\delta+1)$.

Proof. We prove feasibility of the following LP 1.

$$\begin{aligned} (1) \quad & Ax \geq b \\ (2) \quad & x \geq 0 \\ (3) \quad & x \geq x' - \frac{\alpha(1/\delta+1)}{c^T x'} x' \\ (4) \quad & x \leq x' + \frac{\alpha(1/\delta+1)}{c^T x'} x^{OPT} \\ (5) \quad & c^T x \leq (1+\delta)LIN - \alpha \end{aligned} \tag{LP 1}$$

The assumption $\alpha \leq \delta LIN$ implies $c^T x' = (1+\delta)LIN \geq \alpha(1/\delta+1)$ and hence $\frac{\alpha(1/\delta+1)}{c^T x'} \leq 1$. Suppose that LP 1 is feasible and has a solution x''. Due to constraints 3 and 4 the distance between x'' and x' can be bounded. We obtain

$$-\frac{\alpha(1/\delta+1)}{c^T x'} x'_i \leq x''_i - x'_i \leq \frac{\alpha(1/\delta+1)}{c^T x'} x_i^{OPT}$$

and therefore $\|x' - x''\|_1 \leq \alpha(1/\delta+1)(\sum_i \frac{x'_i}{c^T x'} + \sum_i \frac{x_i^{OPT}}{c^T x'}) = \alpha(1/\delta+1)\frac{\|x'\|_1 + \left\|x^{OPT}\right\|_1}{c^T x'}$. If $c^T = (1,1,\ldots,1)$ then $\frac{\|x'\|_1}{c^T x'} = 1$ and $\frac{\left\|x^{OPT}\right\|_1}{c^T x'} = \frac{LIN}{(1+\delta)LIN} = \frac{1}{(1+\delta)} < 1$. Therefore $\alpha(1/\delta+1)\frac{\|x'\|_1 + \left\|x^{OPT}\right\|_1}{c^T x'} < 2\alpha(1/\delta+1)$.

It remains to prove feasibility of LP 1. Therefore, we construct a solution x'' by $x'' = (1 - \frac{\alpha(1/\delta+1)}{c^T x'})x' + \frac{\alpha(1/\delta+1)}{c^T x'} x^{OPT}$. We prove that each constraint of LP 1 is satisfied for x''. Note that constraints 1 and 2 are satisfied since x'' is a convex combination of solutions x' and x^{OPT}. Constraint 3 is fulfilled since $x'' = x' - \frac{\alpha(1/\delta+1)}{c^T x'} x' + \frac{\alpha(1/\delta+1)}{c^T x'} x^{OPT} \geq x' - \frac{\alpha(1/\delta+1)}{c^T x'} x'$ and constraint 4 is fulfilled since $x'' = x' - \frac{\alpha(1/\delta+1)}{c^T x'} x' + \frac{\alpha(1/\delta+1)}{c^T x'} x^{OPT} \leq x' + \frac{\alpha(1/\delta+1)}{c^T x'} x^{OPT}$. Feasibility of constraint 5 is fulfilled as the objective value of x'' is bounded by $c^T x'' = c^T x' - \frac{\alpha(1/\delta+1)}{c^T x'}(c^T x' - c^T x^{OPT}) = c^T x' - \alpha(1/\delta+1) + \frac{\alpha(1/\delta+1)}{(1+\delta)LIN} LIN = c^T x' - \alpha(1/\delta+1) + \alpha(1/\delta) = c^T x' - \alpha$. □

From here on and for the rest of the chapter we suppose that $c^T = (1,1,\ldots,1)$.

In many cases, we do not know the exact approximation ratio $\|x'\|_1 = (1+\delta')LIN$ but the approximation guarantee $\|x'\|_1 \leq (1+\delta)LIN$ for some $\delta \geq \delta'$. Assuming $\|x'\|_1 \geq \alpha(1/\delta+1)$ we can use feasibility of LP 1 to prove the existence of a solution x'' with $\|x''\|_1 \leq (1+\delta)LIN - \alpha$ and $\|x'' - x'\|_1 \leq 2\alpha(1/\delta+1)$.

Theorem 2.2. *Consider the LP $\min\left\{c^T x | Ax \geq b, x \geq 0\right\}$ and an approximate solution x' with $\|x'\| \leq (1+\delta)LIN$ for some $\delta > 0$. For every positive $\alpha \leq \delta LIN$ there exists a solution x'' with objective value of at most $\|x''\| \leq (1+\delta)LIN - \alpha$ and distance $\|x' - x''\|_1 \leq 2\alpha(1/\delta+1)$.*

Proof. Suppose x' has approximation ratio $\|x'\|_1 = (1+\delta')LIN$ for some $0 < \delta' \leq \delta$. By Lemma 2.1, the following LP is feasible for any $\alpha' \leq \delta' LIN$.

$$\begin{aligned} Ax &\geq b \\ x &\geq 0 \\ x &\geq x' - \alpha'(1/\delta'+1)\frac{x'}{\|x'\|_1} \\ x &\leq x' + \alpha'(1/\delta'+1)\frac{x^{OPT}}{\|x'\|_1} \\ \sum x_i &\leq (1+\delta')LIN - \alpha' \end{aligned}$$

Setting $\alpha' = \alpha\frac{(1/\delta+1)}{(1/\delta'+1)}$ for $\delta > 0$ yields feasibility for the following LP assuming $\|x'\|_1 = (1+\delta')LIN$.

$$\begin{aligned} &(1) \quad Ax \geq b \\ &(2) \quad x \geq 0 \\ &(3) \quad x \geq x' - \alpha(1/\delta+1)\frac{x'}{\|x'\|_1} \\ &(4) \quad x \leq x' + \alpha(1/\delta+1)\frac{x^{OPT}}{\|x'\|_1} \\ &(5) \quad \sum x_i \leq (1+\delta')LIN - \alpha' \end{aligned} \qquad \text{(LP *)}$$

Here, we use $\alpha'(1/\delta'+1) = \alpha\frac{1/\delta+1}{1/\delta'+1}(1/\delta'+1) = \alpha(1/\delta+1)$. The condition that $\alpha' \leq \delta' LIN$ is equivalent to the condition that $\|x'\|_1 \geq \alpha(1/\delta+1)$ since $\|x'\|_1 = (1+\delta')LIN$ and $\alpha(1/\delta+1) = \alpha'(1/\delta'+1)$.

The distance $\|x'' - x'\|_1 \leq 2\alpha(1/\delta+1)$ follows as above from constraints 3 and 4 of LP *. We derive the aimed objective value $\|x''\|_1$ from the last constraint of LP * as

$$\begin{aligned} \|x''\|_1 &\leq (1+\delta')LIN - \alpha\frac{1/\delta+1}{1/\delta'+1} = (1+\delta)LIN - (\delta-\delta')LIN - \alpha(1/\delta+1)\frac{1}{1/\delta'+1} \\ &\overset{(1+\delta')LIN \geq \alpha(1/\delta+1)}{\leq} (1+\delta)LIN - \alpha(1/\delta+1)\frac{\delta-\delta'}{1+\delta'} - \alpha(1/\delta+1)\frac{\delta'}{1+\delta'} \\ &= (1+\delta)LIN - \alpha(1/\delta+1)\frac{\delta}{1+\delta'} = (1+\delta)LIN - \alpha\frac{1+\delta}{1+\delta'} \overset{\delta' \leq \delta}{\leq} (1+\delta)LIN - \alpha. \end{aligned}$$

□

Of course, one major application of Theorem 2.1 is to improve the approximation. But we can also apply Theorem 2.1 to obtain a variant of the theorem of Cook et al. [Coo+86] for the sensitivity analysis of an LP. Consider the following problem: Let x' be a solution of $\min\{\|x\|_1 \mid Ax \geq b', x \geq 0\}$. Find a solution x'' for $LIN_2 = \min\{\|x\|_1 \mid Ax \geq b'', x \geq 0\}$ with changed right hand side such that $\|x'' - x'\|_1$ is small. A theorem of Cook et al. [Coo+86] states that there exists a solution x'' satisfying the LP and $\|x'' - x'\|_\infty \leq n\Delta \|b'' - b'\|_\infty$, where Δ is the largest subdeterminant of A. This result is not satisfying if Δ and n are too big, especially if they are exponential in m. By letting loose of optimal solutions we obtain a corollary that is much more appropriate to derive fully robust algorithms. In contrary to the theorem of Cook et al. [Coo+86] the amount of change in the solution does not depend on the determinant nor on the dimensions of A but on the approximation ratio of the solution.

Corollary 2.3. *Consider the linear program LP^1 defined by* $\min\{\|x\|_1 \,|Ax \geq b', x \geq 0\}$ *and the linear program LP^2 defined by* $\min\{\|x\|_1 \,|Ax \geq b'', x \geq 0\}$ *with changed right hand side b''. Let x' be an approximate solution x' of LP^1 with $\|x'\|_1 \leq (1+\delta)LIN_1$ $(0 < \delta \leq 1)$, where LIN_1, LIN_2 is the value of an optimum solution of LP^1 respectively LP^2. Assuming $\|x'\|_1 \geq (1/\delta + 3)\left\|\frac{b''-b'}{v}\right\|_1$, there exists a solution x'' of LP^2 with $\|x''\|_1 \leq (1+\delta)LIN_2$ such that the distance between x'' and x' satisfies $\|x'' - x'\|_1 \leq (\frac{2}{\delta} + 7)\left\|\frac{b''-b'}{v}\right\|_1$ where $v_i = \max_j A_{ij}$ and $\frac{b''-b'}{v}$ is a vector having components $\frac{b''_i-b'_i}{v_i}$.*

Proof. Suppose there is only one index i where $b'_i \neq b''_i$. Consider the 2 cases:
Case 1: $b'_i < b''_i$. We increase x'_j by $\frac{b''_i-b'_i}{v_i}$, where j is the index with the maximum entry in row i. This way we make sure that the so modified x' covers the larger b''_i since now $(Ax')_i = b''_i$. Since we simply increase x' to cover the larger b'' we may worsen the approximation by an additive term of at most $\frac{b''_i-b'_i}{v_i}$.
Case 2: $b'_i \geq b''_i$. In this case we do not modify component i of x', but since a smaller b''_i has to be covered the optimal value of a solution may decrease. The inequality $LIN_2 < LIN_1 - \frac{b''_i-b'_i}{v_i}$ leads to a contradiction since we can increase an optimal solution x^{LIN_1} of LP^1 to cover the larger b' like we did in case 1. Modifying x^{LIN_1} this way would lead to a smaller optimal solution. Therefore the optimal solution of LP^1 can not decline by more than $\frac{b''_i-b'_i}{v_i}$. Using x' as an approximate solution for LP^2 yields therefore $\|x'\|_1 \leq (1+\delta)(LIN_2 + \frac{b''_i-b'_i}{v_i}) = (1+\delta)LIN_2 + (1+\delta)\frac{b''_i-b'_i}{v_i}$.

Iterating over all components $1 \leq i \leq m$ and changing the solution according to the cases would result in an approximate solution of at most $(1+\delta)LIN_2 + (1+\delta)\left\|\frac{b''-b'}{v}\right\|_1$. Using Theorem 2.2 with $\alpha = (1+\delta)\left\|\frac{b''-b'}{v}\right\|_1$ guarantees the existence of a solution $\hat{x}$ for LP^2 having value $(1+\delta)LIN_2$ and $\|\hat{x} - x'\|_1 \leq (2/\delta + 2)\alpha = (2/\delta + 2)(1+\delta)\left\|\frac{b''-b'}{v}\right\|_1 \leq (2/\delta + 6)\left\|\frac{b''-b'}{v}\right\|_1$. □

Note that if A is an integral matrix without zero rows, each component v_i is at least 1.

2.2.1 Algorithmic Use

Let x' be an approximate solution of the LP $\min\{\|x\|_1 \,|Ax \geq b, x \geq 0\}$ with $\|x'\|_1 \leq (1+\delta)LIN$. In Theorem 2.1, we showed the existence of a solution x'' near x' with improved objective value. Given approximate solution x', we are looking now for algorithmic ways to compute the improved solution x''.

The presented algorithms obtain the LP and its approximate LP solution x' as well as α and δ as input. Both algorithms, Algorithm 2.4 and 2.6, compute an approximate solution x'' of the LP with improved objective value $\|x''\| \leq (1+\delta)LIN - \alpha$. Algorithm 2.4 requires to compute an optimal solution $\hat{x}$ of an LP as a subroutine, while Algorithm 2.6 requires only to compute an approximate solution of an LP as a subroutine. The algorithms and theorems in this section are used in the following section to obtain results for integer programming which are then used for the robust bin packing problem.

According to the bounds of LP * we split x' into a fixed part x^{fix} and a variable part x^{var}. The variable part is defined according to LP * by $x^{var} = \frac{\alpha(1/\delta+1)}{\|x'\|}x'$ and the fixed part by $x^{fix} = x' - x^{var}$. We denote with $b^{var} = b - A(x^{fix})$ the part which is not covered by the fixed part of the solution. The following algorithm computes an improved solution

for b^{var} and combines the improved solution with x^{fix}. Assuming $\|x'\|_1 \geq \alpha(1/\delta + 1)$, Theorem 2.2 states that the objective value improves with this algorithm by α.

Algorithm 2.4.

1. *Set* $x^{var} := \frac{\alpha(1/\delta+1)}{\|x'\|} x'$, $x^{fix} := x' - x^{var}$ *and* $b^{var} := b - A(x^{fix})$.
2. *Solve the LP* $\hat{x} = \min\{\|x\|_1 \mid Ax \geq b^{var}, x \geq 0\}$.
3. *Generate a new solution* $x'' = x^{fix} + \hat{x}$.

If $\hat{x}$ is a basic feasible solution, compared to x', our new solution x'' has up to m additional non-zero components. Using the algorithm multiple times yields solutions with more and more non-zero components. This is a problem in the case of bin packing. We address the problem on how to bound the number of non-zero components in the next section.

Theorem 2.5. *Given solution* x' *with* $\|x'\|_1 \leq (1+\delta)LIN$ *and* $\|x'\|_1 \geq \alpha(1/\delta+1)$. *Algorithm 2.4 returns a feasible solution* x'' *with* $\|x''\|_1 \leq (1+\delta)LIN - \alpha$ *and the distance between* x' *and* x'' *is* $\|x'' - x'\|_1 \leq 2\alpha(1/\delta+1)$. *The number of non zero components of* x'' *increases at most by* m *compared to* x'.

Proof. Solution x'' is feasible because $A(x'') = A(x^{fix} + \hat{x}) = A(x^{fix}) + A(\hat{x}) \geq A(x^{fix}) + b^{var} = b$. For the approximation guarantee of x'', we can use Theorem 2.2 and the feasibility of LP LP *. Therefore note that the fixed part x^{fix} is defined by the constraint (3) of LP * which states that $x'' \geq x' - \alpha(1/\delta+1)\frac{x'}{\|x'\|_1}$. In step (2) of the algorithm a solution $\hat{x}$ is computed, which covers the remaining part that is not covered by x^{fix}. Since $\hat{x}$ is computed optimally, $x'' = x^{fix} + \hat{x}$ is a feasible solution of LP LP * and therefore has an objective value of $\leq (1+\delta)LIN - \alpha$.

By definition $\|x^{var}\|_1$ is bounded by $\alpha(1/\delta+1)$ and since $\|\hat{x}\|_1$ is and optimal solution, its is bounded by $\alpha(1/\delta+1)$ as well. Hence the distance between x' and x'' can be bouned by $\|x'' - x'\|_1 = \left\|(x^{fix} + \hat{x}) - (x^{fix} + x^{var})\right\|_1 = \|\hat{x} - x^{var}\|_1 \leq \|\hat{x}\|_1 + \|x^{var}\|_1 \leq 2\alpha(1/\delta+1)$. □

In Algorithm 2.4 we use an optimal LP solver as a subroutine. In many cases, like for example bin packing, the corresponding LP relaxation is hard to solve and the running time for computing an optimal solution is very high. For the following algorithm it is sufficient to compute the LP approximately, which in general can be performed more efficiently. We assume that $\|x'\|_1 \geq 2\alpha(1/\delta+1)$ because the double amount has to be reassigned to achieve the same improvement in the approximation as in Algorithm 2.4.

Algorithm 2.6.

1. *Set* $x^{var} := \frac{2\alpha(1/\delta+1)}{\|x'\|} x'$, $x^{fix} := x' - x^{var}$ *and* $b^{var} := b - A(x^{fix})$.
2. *Solve* $\hat{x} = \min\{\|x\|_1 \mid Ax \geq b^{var}, x \geq 0\}$ *approximately with ratio* $(1+\delta/2)$.
3. *If* $\left\|x^{fix} + \hat{x}\right\|_1 < \|x'\|_1$ *set* $x'' := x^{fix} + \hat{x}$ *else* $x'' = x'$.

Theorem 2.7. *Let* x' *be a solution with* $\|x'\|_1 \leq (1+\delta)LIN$ *and* $\|x'\|_1 \geq 2\alpha(1/\delta+1)$. *Then Algorithm 2.6 returns a feasible solution* x'' *with approximation guarantee* $(1+\delta)LIN - \alpha$ *and* $\|x'' - x'\|_1 \leq 4\alpha(1/\delta+1)$.

Proof. The property that $\|x'' - x'\|_1 \leq 4\alpha(1/\delta + 1)$ follows by the same proof argument as in Theorem 2.5 and the fact that x^{var} has the double size $2\alpha(1/\delta+1)$ compared to x^{var} defined in Algorithm 2.4. It remains to show that at the end of the algorithm x'' achieves the aimed approximation guarantee $\|x''\|_1 \leq (1+\delta)LIN - \alpha$. Suppose $\|x'\| = (1+\delta')LIN$ for some $\delta' \leq \delta$. Using the assumption $2\alpha(1/\delta+1) \leq \|x'\|_1 \leq (1+\delta)LIN$ implies that $2\alpha \leq \frac{(1+\delta)LIN}{(1/\delta+1)} = \delta LIN$. Consider the case that $2\delta' \leq \delta$. In this case x'' has the aimed approximation since $\|x''\|_1 \leq \|x'\|_1 = (1+\delta')LIN \leq (1+\delta)LIN - \delta/2LIN \leq (1+\delta)LIN - \alpha$ using $2\alpha \leq \delta LIN$. Thus in the following we assume $\delta \leq 2\delta'$. Suppose we solve the LP in step 2 optimally. In this case, Algorithm 2.6 is identical to Algorithm 2.4 using an improvement of 2α. By feasibility of LP * we know there exists a solution $\bar{x}''$ with $\|\bar{x}''\| \leq (1+\delta')LIN - 2\alpha(\frac{1/\delta+1}{1/\delta'+1})$. This implies that an optimal solution $\hat{x}^{OPT}$ of the LP $\min\{\|x\|_1 \mid Ax \geq b^{var}, x \geq 0\}$ is of size $\left\|\hat{x}^{OPT}\right\|_1 \leq \|x^{var}\|_1 - 2\alpha\frac{1/\delta+1}{1/\delta'+1} = 2\alpha(1/\delta + 1) - 2\alpha\frac{1/\delta+1}{1/\delta'+1}$. Solving the LP approximately with ratio $(1+\delta/2)$, the objective value of $\hat{x}$ has an additional term $\delta/2\left\|\hat{x}^{OPT}\right\|_1$. The value of $\|\hat{x}\|_1$ is therefore bounded by $\|\hat{x}\|_1 \leq \left\|\hat{x}^{OPT}\right\|_1 + \delta/2\left\|\hat{x}^{OPT}\right\|_1 \leq \|x^{var}\|_1 - 2\alpha\frac{1/\delta+1}{1/\delta'+1} + \alpha(1+\delta) - \alpha(\frac{1+\delta}{1/\delta'+1})$. Finally this results in the approximation for $x^{fix} + \hat{x}$ as follows.

$$
\begin{aligned}
\left\|x^{fix} + \hat{x}\right\|_1 &= \|x'\|_1 - \|x^{var}\|_1 + \|\hat{x}\|_1 \\
&= (1+\delta')LIN - 2\alpha(\frac{1/\delta+1}{1/\delta'+1}) + \alpha(1+\delta) - \alpha(\frac{1+\delta}{1/\delta'+1}) \\
&= (1+\delta)LIN - (\delta-\delta')LIN - 2\alpha(\frac{1/\delta+1}{1/\delta'+1}) + \alpha(1+\delta) - \alpha(\frac{1+\delta}{1/\delta'+1}) \\
&\overset{LIN \geq 2\alpha/\delta}{\leq} (1+\delta)LIN - 2\alpha(\frac{\delta-\delta'}{\delta}) + \alpha(1+\delta) - 2\alpha(\frac{1/\delta+1}{1/\delta'+1}) - \alpha(\frac{1+\delta}{1/\delta'+1}) \\
&= (1+\delta)LIN + \alpha(\frac{-2\delta+2\delta'+\delta+\delta^2}{\delta}) - 2\alpha(\frac{1/\delta+1}{1/\delta'+1}) - \alpha(\frac{1+\delta}{1/\delta'+1}) \\
&= (1+\delta)LIN - \alpha + \alpha(\frac{2\delta'+\delta^2}{\delta}) - 2\alpha(\frac{1/\delta+1}{1/\delta'+1}) - \alpha(\frac{1+\delta}{1/\delta'+1}) \\
&= (1+\delta)LIN - \alpha + \alpha(\frac{2+2\delta'+\delta^2/\delta'+\delta^2-2-2\delta-\delta-\delta^2}{\delta(1/\delta'+1)}) \\
&= (1+\delta)LIN - \alpha + \alpha(\frac{2\delta'+\delta^2/\delta'-3\delta}{\delta(1/\delta'+1)}) \\
&= (1+\delta)LIN - \alpha + \alpha(\frac{(\delta-\delta')(-2+\frac{\delta}{\delta'})}{\delta(1/\delta'+1)}) \\
&\leq (1+\delta)LIN - \alpha
\end{aligned}
$$

The last inequality holds because $\alpha(\frac{(\delta-\delta')(-2+\frac{\delta}{\delta'})}{\delta(1/\delta'+1)}) \leq 0$ since $\delta - \delta' \geq 0$ and $-2 + \frac{\delta}{\delta'} \leq 0 \Leftrightarrow \delta \leq 2\delta'$. By the last step of the algorithm we know that $\|x''\|_1 \leq \left\|x^{fix} + \hat{x}\right\|_1$ and thus $\|x''\|_1 \leq (1+\delta)LIN - \alpha$. □

In some cases we may not want to get a guaranteed approximation, but a guarantee that our solution x' is getting smaller by some α. This works if the approximation ratio of x' is worse than $1+\delta$. The following corollary states, that if we use Algorithm 2.6 on a solution x' with $\|x'\|_1 = (1+\delta')LIN$ for some $\delta' \geq \delta$ the objective function of our new solution x'' decreases by at least α.

Corollary 2.8. *Let* $\|x'\|_1 = (1+\delta')LIN$ *for some* $\delta' \geq \delta$ *and* $\|x'\|_1 \geq 2\alpha(1/\delta+1)$*. Then Algorithm 2.6 returns a solution* x'' *with* $\|x''\|_1 \leq \|x'\|_1 - \alpha = (1+\delta')LIN - \alpha$ *and* $\|x''-x'\|_1 \leq 4\alpha(1/\delta+1)$*.*

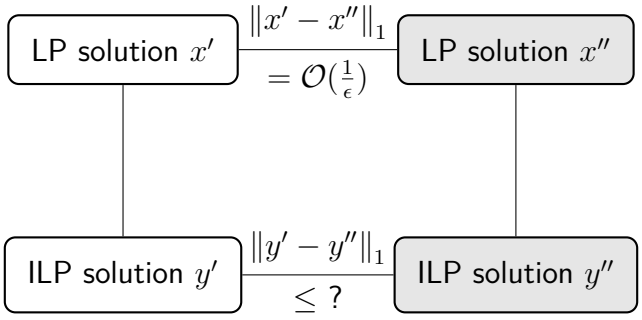
Proof. Suppose like in the proof of Theorem 4 that we solve the LP in step 2 optimally. In this case, Algorithm 2.6 is identical to Algorithm 2.4 using improvement of 2α and therefore by feasibility of LP * we know that it returns a solution $\bar{x}''$ with $\|\bar{x}''\| \leq (1+\delta')LIN - 2\alpha(\frac{1/\delta+1}{1/\delta'+1})$. An optimal solution $\hat{x}^{OPT}$ of the LP $\min\{\|x\|_1 \mid Ax \geq b^{var}, x \geq 0\}$ is therefore of size $\left\|\hat{x}^{OPT}\right\|_1 \leq \|x^{var}\|_1 - 2\alpha\frac{1/\delta+1}{1/\delta'+1} = 2\alpha(1/\delta+1) - 2\alpha\frac{1/\delta+1}{1/\delta'+1}$. Since we actually solve the LP approximately with ratio $(1+\delta/2)$, solution $\hat{x}$ has an additional term of $\delta/2\left\|\hat{x}^{OPT}\right\|_1$ and the value is therefore bounded by $\|\hat{x}\|_1 \leq \|x^{var}\|_1 - 2\alpha\frac{1/\delta+1}{1/\delta'+1} + \alpha(1+\delta) - \alpha(\frac{1+\delta}{1/\delta'+1})$ according to the proof of Theorem 5. By construction of x'' we get $\|x''\|_1 \leq \left\|x^{fix}+\hat{x}\right\|_1 = (1+\delta')LIN - 2\alpha(\frac{1/\delta+1}{1/\delta'+1}) + \alpha(1+\delta) - \alpha(\frac{1+\delta}{1/\delta'+1})$. Since $\delta' \geq \delta$ we know that $-2\alpha(\frac{1/\delta+1}{1/\delta'+1}) \leq -2\alpha$ and that $\alpha(1+\delta) - \alpha(\frac{1+\delta}{1/\delta'+1}) \leq \alpha$. Hence $\|x''\|_1 \leq (1+\delta')LIN - \alpha$. □

2.3 Integer Programming

In this section we discuss how the methods from the previous section can be applied to integer programming. We consider pairs (x', y') of given fractional solution x' and integer solution y' with $y'_i \geq x'_i$ for every index i. We say the integral solution y' *corresponds* to the fractional solution x'. The approximation guarantee of the integral solution y' may involve some additive term $C \in \mathbb{R}^+$ i.e. $\|y'\|_1 \leq (1+\delta)LIN + C$. Given a fractional solution x': By rounding each component x'_i up, it is easy to get a feasible integer solution y' with an additional additive term $\|y'\|_1 \leq \|x'\|_1 + C$, where C is the number of non-zero components of x'.

Given a pair (x', y') with approximation guarantee $\|x'\|_1 \leq (1+\delta)LIN$ and $\|y'\|_1 \leq (1+\delta)LIN + C$ for some $C \in \mathbb{R}^+$, our goal is to find a pair (x'', y'') with improved objective value $\|y''\|_1 \leq (1+\delta)LIN + C - \alpha$ such that the distance between y' and y'' is bounded. In this section we develop three algorithms to achieve this goal. Each algorithm obtains x', y', α, δ and the LP as input. The first Algorithm 2.9 requires to reduce y' by at most $\alpha(1/\delta+2)$ while the additive term C is bounded by n. The second algorithm achieves $\|y''-y'\| \leq \mathcal{O}(\frac{m+\alpha}{\delta})$ while the additive term is bounded by δLIN. Finally, we will give Algorithm 2.13, which is a modification of Algorithm 2.11. Algorithm 2.13 will be used in Section 2.4 for to obtain the robust AFPTAS for the bin packing problem. Each algorithm builds upon the ideas from the previous one.

We will use methods from Section 2 to obtain an improved fractional solution x'' from x'. Note that the straight forward approach to simply round up each component x''_i leads to a distance between y'' and y' that depends on the number of non-zero components and hence is too high.



Like in the previous algorithms of Section 2, we assume that $\|x'\|_1 \geq \alpha(1/\delta + 1)$. We require x' to be a solution with approximation guarantee $\|x'\|_1 \leq (1+\delta)LIN$ and we require y' to be an integer solution with approximation guarantee $\|y'\|_1 \leq (1+\delta)LIN + n$. For every $1 \leq i \leq n$ we suppose that $x'_i \leq y'_i$. For a vector $z \in \mathbb{R}^n_{\geq 0}$, let $V(z)$ be the set of all integral vectors $v = (v_1, \ldots, v_n)^T$ such that $0 \leq v_i \leq z_i$. Given LP solution x' and integer solution y' with the described properties, the algorithm performs in the following way:

Algorithm 2.9.

1. *If possible choose vector $c \in V(y' - x')$ with $\|c\|_1 = \alpha$ and return $y'' = y' - c$ and $x'' = x'$. Otherwise choose $c \in V(y' - x')$ such that $\|c\|_1 < \alpha$ is maximized. Set $\bar{y} = y' - c$.*
2. *Set $x^{var} := \frac{\alpha(1/\delta+1)}{\|x'\|} x'$, $x^{fix} := x' - x^{var}$ and $b^{var} := b - A(x^{fix})$.*
3. *Compute an optimal solution $\hat{x}$ of the LP $\min\{\|x\|_1 \mid Ax \geq b^{var}, x \geq 0\}$.*
4. *Set $x'' = x^{fix} + \hat{x}$.*
5. *For each $1 \leq i \leq n$ set $\hat{y}_i = \max\{\lceil x''_i \rceil, \bar{y}_i\}$.*
6. *If possible choose $d \in V(\hat{y} - x'')$ such that $\|d\|_1 = \alpha(1/\delta + 1)$ otherwise choose $d \in V(\hat{y} - x'')$ such that $\|d\|_1 < \alpha(1/\delta + 1)$ is maximized.*
7. *Return $y'' = \hat{y} - d$.*

Explanation of the algorithm:
In step 1, the algorithm decreases components y'_i with $y'_i - x'_i \geq 1$. The fractional solution x'' is defined in steps 2-4, which are identical to Algorithm 2.4. In steps 6,7 the components of $\hat{y}$ are decreased in the same way as in step 1. The main idea of the algorithm lies in the distinction of the cases between $\|d\|_1 < \alpha(1/\delta + 1)$ and $\|d\|_1 = \alpha(1/\delta + 1)$. In the case that $\|d\|_1 < \alpha(1/\delta + 1)$ the fractional solution x'' is sufficiently close to y'' to guarantee a good approximation guarantee for y'' and in the case that $\|d\|_1 = \alpha(1/\delta + 1)$ the integral solution y'' has good approximation guarantee by comparing it to y'.

Theorem 2.10. *Let x' be a solution of the LP with $\|x'\|_1 \leq (1+\delta)LIN$ and $\|x'\|_1 \geq \alpha(1/\delta + 1)$. Let y' be an integral solution of the LP with $\|y'\|_1 \leq (1+\delta)LIN + n$ where $y'_i \geq x'_i$ for each $i = 1, \ldots, n$. Then Algorithm 2.9 returns an integral solution y'' with $\|y''\|_1 \leq (1+\delta)LIN + n - \alpha$ such that $\sum_{y''_i < y'_i}(y'_i - y''_i) \leq \alpha(1/\delta + 2)$.*

Note that $\sum_{y''_i < y'_i}(y'_i - y''_i)$ denotes the total sum where components of y' need to be reduced to obtain y''. This term determines the migration factor in Section 2.4 for the bin packing problem.

Proof. **Feasibility:** Feasibility for x'' and approximation $\|x''\|_1 \leq (1+\delta)LIN - \alpha$ follows from Theorem 3. Step 2,3 and 4 are identical to Algorithm 2.4. Feasibility for the integer solution y'' follows from the fact, that for every component i we have $y''_i = \hat{y}_i - d_i \geq x''_i$ and hence $Ay'' \geq Ax'' \geq b$.
Size of reduction of y': The only steps where components of y' are changed are in step 1, 5 and 6. In step 1 we change y' to obtain $\bar{y}$, in step 5 we change $\bar{y}$ to obtain $\hat{y}$ and in step 6 we change $\hat{y}$ to obtain y''. Summing up the change in each step leads therefore to the maximum possible size of reduction of y' compared to y''. In step 1 there are $c \leq \alpha$ components of y' which are being reduced. In step 5 no components of $\bar{y}$ are being reduced and in step 6 there are $d \leq \alpha(1/\delta + 1)$ components of $\hat{y}$ which are being reduced to obtain y''. Hence there are at most $\alpha(1/\delta + 2)$ components of y' which are being reduced to obtain y''.
Approximation: It remains to prove that y'' has approximation ratio $(1+\delta)LIN + n - \alpha$. Case 1, $\|c\|_1 = \alpha$: In this case, the algorithm returns in step 1 solution $y'' = \bar{y}$ with $\|y''\|_1 = \|y'\|_1 - \alpha$ and the algorithm terminates. Case 2, $\|c\|_1 < \alpha$: As c is chose maximally, we have for every component i, that $\bar{y}_i - x'_i < 1$ since c could be increased otherwise. Furthermore, we know that $\|x''\|_1 \leq (1+\delta)LIN - \alpha$ as steps 2-4 are equivalent to Algorithm 2.4.
Case 2.1, $\|c\|_1 < \alpha$ and $\|d\|_1 < \alpha(1/\delta + 1)$: In this case $y''_i - x''_i = \hat{y}_i - d_i - x''_i < 1$ for $i = 1, \ldots, n$, since $\|d\|_1$ is chosen maximally. Using $\|x''\|_1 \leq (1+\delta)LIN - \alpha$ and $y''_i < x''_i + 1$ for $i = 1, \ldots, n$ we have $\|y''\|_1 \leq (1+\delta)LIN + n - \alpha$.
Case 2.2, $\|c\|_1 < \alpha$ and $\|d\|_1 = \alpha(1/\delta + 1)$: Let $\bar{m}$ be the number of components with $x''_i > \bar{y}_i$. Next we compare the vector $\hat{y}$ with x''. Using $x'' \geq x^{fix}$ and the definition of $\hat{y}$ in step 5 we obtain $\|\hat{y} - x''\|_1 = \sum_{x''_i \leq \bar{y}_i} (\bar{y}_i - x''_i) + \sum_{x''_i > \bar{y}_i} (\lceil x''_i \rceil - x''_i) \leq \sum_{x''_i \leq \bar{y}_i} (\bar{y}_i - x^{fix}_i) + \bar{m}$. The fact that $\bar{y}_i - x'_i < 1$ for $i = 1, \ldots, n$ and $\|x'\|_1 - \left\|x^{fix}\right\|_1 = \|x^{var}\|_1 \leq \alpha(1/\delta + 1)$ and the fact there are at most $n - \bar{m}$ components with $x''_i < \bar{y}_i$ yield that $\sum_{x''_i \leq \bar{y}_i} (\bar{y}_i - x^{fix}_i) = \sum_{x''_i \leq \bar{y}_i} (\bar{y}_i - x'_i + x^{var}_i) \leq n - \bar{m} + \sum_{x''_i \leq y'_i} x^{var}_i \leq n - \bar{m} + \alpha(1/\delta + 1)$. As a result we can bound $\|\hat{y} - x''\|_1 \leq \sum_{x''_i \leq \bar{y}_i} (\bar{y}_i - x^{fix}_i) + \bar{m} \leq n + \alpha(1/\delta + 1)$. Since $y'' = \hat{y} - d$ and $\|d\|_1 = \alpha(1/\delta + 1)$, our integer solution y'' has the aimed approximation guarantee of

$$\|y''\|_1 = \|\hat{y}\|_1 - \|d\|_1 \leq \|x''\| + \alpha(1/\delta + 1) + n - \|d\|_1 = \|x''\| + n \leq (1+\delta)LIN + n - \alpha.$$

□

The running time of the above algorithm depends on the number of non-zero components and the time to compute an optimal solution of an LP. The algorithms computes an integral solution y'' with $\|y''\|_1 \leq (1+\delta)LIN + n - \alpha$ for given fractional and integral solution. In many cases, like bin packing, the dimension n is very large and provides thus a large additive term in the approximation. The following algorithm describes how this large additive term can be avoided. On the other hand the difference between y' and y'' increases to $\mathcal{O}(\frac{m+\alpha}{\delta})$. Let x' be an approximate solution of the LP $\min\{\|x\|_1 \mid Ax \geq b, x \geq 0\}$ with $\|x'\|_1 \leq (1+\delta)LIN$ and $\|x'\|_1 \geq \alpha(1/\delta + 1)$. Furthermore let y' be an approximate integer solution of the LP with $\|y'\|_1 \leq (1+2\delta)LIN$ and $\|y'\|_1 \geq (m+1)(1/\delta + 2)$ and $y'_i \geq x'_i$ for $i = 1, \ldots, n$. In addition we suppose that both x' and y' have exactly $K \leq \delta LIN$ non-zero components. Our goal is now to compute a fractional solution x'' and and integer solution y'' having improved approximation properties and still $\leq \delta LIN$ non-zero components. Furthermore we denote with $a_1, \ldots, a_K$ the indices of the non-zero components y'_{a_j} such that $y'_{a_1} \leq \ldots \leq y'_{a_K}$ are sorted in non-decreasing order.

Algorithm 2.11.

1. *Choose ℓ maximally such that the sum of smallest ℓ components $y'_{a_1}, \ldots, y'_{a_\ell}$ is $\sum_{1 \le i \le \ell} y'_{a_i} \le (m+1)(1/\delta + 2)$.*
2. *Set* $x_i^{var} = \begin{cases} x'_i & \text{if } i = a_j \text{ for } j \le \ell \\ \frac{\alpha(1/\delta+1)}{\|x'\|} x'_i & \text{else} \end{cases}$ *and* $\bar{y}_i = \begin{cases} 0 & \text{if } i = a_j \text{ for } j \le \ell \\ y'_i & \text{else} \end{cases}$.
3. *Set $x^{fix} = x' - x^{var}$, $b^{var} = b - A(x^{fix})$ and compute an optimal solution $\hat{x}$ of the LP $\min\{\|x\|_1 \mid Ax \ge b^{var}, x \ge 0\}$.*
4. *Set $x'' = x^{fix} + \hat{x}$.*
5. *For each $1 \le i \le n$ set $\hat{y}_i = \max\{\lceil x''_i \rceil, \bar{y}_i\}$*
6. *If possible choose $d \in V(\hat{y} - x'')$ such that $\|d\|_1 = \alpha(1/\delta + 1)$ otherwise choose $d \in V(\hat{y} - x'')$ such that $\|d\|_1 < \alpha(1/\delta + 1)$ is maximized.*
7. *Return $y'' = \hat{y} - d$.*

Explanation of the algorithm:
The main ingredient of this algorithm is to reassign the smallest components $y_{a_1}, \ldots, y_{a_\ell}$ in order to make sure that the total number of non-zero components of x'' and y'' does not exceed δLIN. Those components are set to 0 in step 2 and then reassigned by the LP in step 3. The remaining part of the algorithm is analogue to Algorithm 2.9.

Theorem 2.12. *Let x' be a solution of the LP with $\|x'\|_1 \le (1+\delta)LIN$ and $\|x'\|_1 \ge \alpha(1/\delta + 1)$. Let y' be an integral solution of the LP with $\|y'\|_1 \le (1+2\delta)LIN$ and $\|y'\|_1 \ge (m+1)(1/\delta+2)$. Solutions x' and y' have both exactly K non-zero components and for each component we have $x'_i \le y'_i$. Then Algorithm 2.11 returns a fractional solution x'' with $\|x''\|_1 \le (1+\delta)LIN - \alpha$ and an integral solution y'' with $\|y''\|_1 \le (1+2\delta)LIN - \alpha$. Both x'' and y'' have the same number of non-zero components with $x''_i \le y''_i$ and the number of non-zero components is bounded by δLIN. The distance between y'' and y' is bounded by $\|y'' - y'\|_1 = \mathcal{O}(\frac{m+\alpha}{\delta})$.*

Proof. **Feasibility**: Feasibility and approximation for the fractional solution x'' follow easily from correctness of Algorithm 2.4 and the fact that removing additional components $x'_{a_1}, \ldots, x'_{a_\ell}$ and reassigning them optimally does not worsen the approximation. Each integral component $\hat{y}_i$ is by definition (step 5) greater or equal than x''_i. By choice of d step 6 and 7 retain this property for y'' and imply thus feasibility for y''.

Distance between y'' and y': The only steps where components of y' are changed are step 2, 5 and 7. In step 2 we change y' to obtain $\bar{y}$, in step 5 we change $\bar{y}$ to obtain $\hat{y}$ and in step 7 we change $\hat{y}$ to obtain y''. Summing up the change in each step leads therefore to the maximum possible distance between y'' and y'. In step 2 of the algorithm ℓ components of y' are set to zero to obtain $\bar{y}$, which by the definition of ℓ results in a change of at most $(m+1)(1/\delta+2)$. We define L by $L = \sum_{1 \le i \le \ell} y'_{a_i}$ with $0 \le L \le (m+1)(1/\delta+2)$. In step 5, the only components $\bar{y}_i$ being changed are the ones where x''_i is larger than $\bar{y}_i$. So the change in step 5 is bounded by $\sum_{x''_i > \bar{y}_i}(\lceil x''_i \rceil - \bar{y}_i) = \sum_{x''_i > \bar{y}_i}(\lceil x_i^{fix} + \hat{x}_i \rceil - \bar{y}_i) \le \sum_{x''_i > \bar{y}_i}(\lceil x_i^{fix} \rceil - \bar{y}_i + \lceil \hat{x}_i \rceil) \le \sum_{x''_i > \bar{y}_i} \lceil \hat{x}_i \rceil$ by knowing that $\lceil x_i^{fix} \rceil - \bar{y}_i \le 0$ since $x_i^{fix} = \bar{y}_i = 0$ if $i = a_j$ for a $j \le \ell$ or $\lceil x_i^{fix} \rceil < \lceil x'_i \rceil \le y'_i$.

Furthermore we can bound $\sum_{x''_i > \bar{y}_i} \lceil \hat{x}_i \rceil \leq \|\hat{x}\|_1 + m \leq \|x^{var}\|_1 + m$ using that $\hat{x}$ is an optimal basic feasible solution. Now, $\|x^{var}\|_1$ can be bounded by $L + \alpha(1/\delta + 1)$ (i.e. we get L for the size of components $x'_{a_1}, \ldots, x'_{a_K}$ plus $\sum_{i>\ell} \frac{\alpha(1/\delta+1)}{\|x'\|_1} x'_{a_i} \leq \alpha(1/\delta + 1)$ for the remaining ones). Therefore, we have $\|\hat{y} - \bar{y}\|_1 \leq L + \alpha(1/\delta + 1) + m$. In step 7, $\|y'' - \hat{y}\|_1 = \|d\|_1 \leq \alpha(1/\delta + 2) + m$. In sum this makes a total change of at most

$$\begin{aligned}\|y'' - y'\|_1 &\leq (m+1)(1/\delta + 2) + L + \alpha(1/\delta + 1) + m + \alpha(1/\delta + 2) + m \\ &\leq 2(m+1)(1/\delta + 2) + 2m + \alpha(2/\delta + 3) = \mathcal{O}(\frac{m+\alpha}{\delta}).\end{aligned}$$

Number of components: The property that x' and y' have the same number of non-zero components together with the property that $y'_i \geq x'_i$ implies that $x'_i > 0$ whenever $y'_i > 0$. This property holds also for x^{fix} and $\bar{y}$ since a component $\bar{y}_i$ is set to zero if and only if $x_i^{fix} = 0$. Notice that $y'' = \hat{y} - d \geq x''$. Suppose by contradiction that there is a component i with $x''_i = 0$ and $y''_i > 0$, then $\hat{y}_i = y''_i + d_i > 0$ and by definition of $\hat{y}$ we obtain $\bar{y}_i > 0$. In this case we have $x_i^{fix} > 0$, which gives a contradiction to $x''_i = 0 = x_i^{fix} + \hat{x}_i > 0$. Using the property that x'' and y'' have the same number of non-zero components, it is sufficient to prove that the number of non-zero components of x'' is limited by δLIN. Our new solution x'' is composed of x^{fix} and $\hat{x}$. Solution x^{fix} has $K - \ell$ non-zero components, since in step 2 we set ℓ components of x^{fix} to zero. Being a basic feasible solution, $\hat{x}$ has at most m non-zero components and hence x'' has at most $K + m - \ell$ non-zero components. If $\ell \geq m$, then x'' has $\leq K \leq \delta LIN$ non-zero components. So let $\ell < m$: The total number of non-zero components after step 4 is $(K + m - \ell)$. We now prove that this number is bounded by δLIN. Parameter ℓ is chosen to be maximal, therefore $\sum_{i \leq l+1} y'_{a_i} \geq (m+1)(1/\delta + 2)$. Hence, the average size of components $y'_{a_1}, \ldots, y'_{a_{\ell+1}}$ is greater than $\frac{(m+1)(1/\delta+2)}{\ell+1} \overset{\ell+1 \leq m}{\geq} \frac{(m+1)(1/\delta+2)}{m} > 1/\delta + 2$. Since the components are sorted in non-decreasing order, every component y'_i with $i \geq \ell + 1$ has size $> 1/\delta + 2$. Summing over all non-zero components of y' yields the following inequality:

$$\begin{aligned}\|y'\|_1 &= \sum_{i=\ell+2}^{K} y'_{a_i} + y'_{a_{\ell+1}} + L \geq (K - \ell - 1)(1/\delta + 2) + y'_{a_{\ell+1}} + L \\ &\geq (K - \ell - 1)(1/\delta + 2) + (m+1)(1/\delta + 2) = (K - \ell + m)(1/\delta + 2).\end{aligned}$$

Using that $\|y'\|_1 \leq (1 + 2\delta)LIN$ yields $(1 + 2\delta)LIN \geq (K - \ell + m)(1/\delta + 2)$. Dividing both sides by $(1/\delta + 2)$ gives $(K - \ell + m) \leq \delta LIN$. This shows that the number of non-zero components of x'' and y'' is at most δLIN.

Approximation: Case1: $\|d\|_1 = \alpha(1/\delta + 2) + m$
The following inequalities $\|\hat{y}\|_1 \leq \|\bar{y}\|_1 + L + \alpha(1/\delta + 2) + m = \|y'\|_1 + \alpha(1/\delta + 1) + m$ and $\|y'\|_1 \leq (1 + 2\delta)LIN$ together yield the aimed approximation $\|y''\|_1 = \|\hat{y}\|_1 - \|d\|_1 = \|\hat{y}\|_1 - \alpha(1/\delta + 2) - m \leq (1 + 2\delta)LIN - \alpha$.
Case2: $\|d\|_1 < \alpha(1/\delta + 2) + m$
Since d is chosen maximally, $y''_i - x''_i < 1$ for every components $i = 1, \ldots, n$. Since $\|x''\|_1 \leq (1 + \delta)LIN - \alpha$ and y'' has at most δLIN non-zero components $\|y''\|_1$ is bounded by $(1 + \delta)LIN - \alpha + \delta LIN = (1 + 2\delta)LIN - \alpha$. □

Instead of using an optimal LP solution in Algorithm 2.9 and 2.11, we can solve the LP approximately with a ratio of $(1 + \delta/2)$. The following algorithm is basically a combination of Algorithm 2.6 and Algorithm 2.11. We could also combine Algorithm 2.6 and Algorithm

2.9 to obtain similar results. We make the following assumption for the fractional solution x' and the corresponding integer solution y': Let x' be an approximate solution of the LP $\min\{\|x\|_1 \mid Ax \geq b, x \geq 0\}$ with $\|x'\|_1 \leq (1+\delta)LIN$ and $\|x'\|_1 \geq 2\alpha(1/\delta+1)$. Let y' be an approximate integer solution of the LP with $\|y'\|_1 \leq LIN + 2C$ for some value $C \geq \delta LIN$ and with $\|y'\|_1 \geq (m+2)(1/\delta+2)$. Suppose that both x' and y' have only $K \leq C$ non-zero components. For every component i we suppose that $y'_i \geq x'_i$. Furthermore we are given indices $a_1, \ldots, a_K$, such that the non-zero components y'_{a_j} are sorted in non-decreasing order i.e. $y'_{a_1} \leq \ldots \leq y'_{a_K}$.

Algorithm 2.13.

1. *Set* $x^{var} := 2\frac{\alpha(1/\delta+1)}{\|x'\|}x'$, $x^{fix} := x' - x^{var}$ *and* $b^{var} = b - A(x^{fix})$.
2. *Compute an approximate solution* $\hat{x}$ *of the LP* $\min\{\|x\|_1 \mid Ax \geq b^{var}, x \geq 0\}$ *with ratio* $(1+\delta/2)$.
3. *If* $\left\|x^{fix} + \hat{x}\right\|_1 \geq \|x'\|_1$ *then set* $x'' = x'$, $\hat{y} = y'$ *and goto step 9*
4. *Choose the largest* ℓ *such that the sum of smallest components* $y'_{a_1}, \ldots, y'_{a_\ell}$ *is* $\sum_{1 \leq i \leq \ell} y'_{a_i} \leq (m+2)(1/\delta+2)$.
5. *For all* i *set* $\bar{x}_i^{fix} = \begin{cases} 0 & \text{if } i = a_j \text{ for } j \leq \ell \\ x_i^{fix} & \text{else} \end{cases}$ *and* $\bar{y}_i = \begin{cases} 0 & \text{if } i = a_j \text{ for } j \leq \ell \\ y'_i & \text{else} \end{cases}$.
6. *Set* $\bar{x} = \hat{x} + x'_\ell$ *where* x'_ℓ *is a vector consisting of components* $x_{a_1}, \ldots, x_{a_\ell}$. *Reduce the number of non-zero components to at most* $m+1$.
7. $x'' = \bar{x}^{fix} + \bar{x}$.
8. *For all non-zero components* i *set* $\hat{y}_i = \max\{\lceil x''_i \rceil, \bar{y}_i\}$
9. *If possible choose* $d \in V(\hat{y} - x'')$ *such that* $\|d\|_1 = \alpha(1/\delta+1)$ *otherwise choose* $d \in V(\hat{y} - x'')$ *such that* $\|d\|_1 < \alpha(1/\delta+1)$ *is maximized.*
10. *Return* $y'' = \hat{y} - d$.

Step 6 of the algorithm can be performed using a standard technique presented for example in [BM98]. Arbitrary many components of $\bar{x}$ can be reduced to $m+1$ many without making the approximation guarantee worse.

Explanation of the algorithm:
Since the algorithm solves the LP in step 2 approximately, some adjustments to Algorithm 2.11 had to be made. In step 3 of the algorithm it is tested if the solution $x^{fix} + \hat{x}$ actually improves upon the old solution x'. The smallest components $y_{a_1}, \ldots, y_{a_\ell}$ are defined in step 4. They are reduced together with $\hat{x}$ in step 6.

The following theorem and corollary is stated in a way such that it can be directly used in the next section.

Theorem 2.14. *Let* x' *be a solution of the LP with* $\|x'\|_1 \leq (1+\delta)LIN$ *and* $\|x'\|_1 \geq 2\alpha(1/\delta+1)$. *Let* y' *be an integral solution of the LP with* $\|y'\|_1 \leq LIN + 2C$ *for some value* $C \geq \delta LIN$ *and with* $\|y'\|_1 \geq (m+2)(1/\delta+2)$. *Solutions* x' *and* y' *have the same number of non-zero components and for each component we have* $x'_i \leq y'_i$. *The number of non-zero components of* x' *and* y' *is* K *with* $K \leq C$. *Then Algorithm 2.13 returns a*

fractional solution x'' with $\|x''\|_1 \leq (1+\delta)LIN - \alpha$ and an integral solution y'' where one of the two properties hold: $\|y''\|_1 = \|y'\|_1 - \alpha$ or $\|y''\|_1 = \|x''\|_1 + C$. Both, x'' and y'' have at most C non-zero components and the distance between y'' and y' is bounded by $\|y'' - y'\|_1 = \mathcal{O}(\frac{m+\alpha}{\delta})$.

Proof. Note that the first 3 steps are equivalent to Algorithm 2.6. In steps 4-6 the number of non-zero components $x'_{a_1}, \ldots, x'_{a_\ell}$ are reduced. As we apply a method that does not increase the objective value we obtain by Theorem 4 that $\|x''\|_1 \leq (1+\delta)LIN - \alpha$. Steps 4-9 are similar to Algorithm 2.11. The main difference is that components $x'_{a_1}, \ldots, x'_{a_\ell}$ are not assigned by the LP but are added to the LP solution afterwards in step 7.
Distance between y'' and y': As in Theorem 7, the steps where components of y' are changed are steps 5,8 and 10. By definition of ℓ the change of y' in step 5 is bounded by $(m+2)(1/\delta+2)$. As shown, the change in step 8 is bounded by $\sum_{x''_i > \bar{y}_i} \lceil x_i^{fix} + \hat{x}_i \rceil - \bar{y}_i) \leq \sum_{x''_i > \bar{y}_i} \lceil \bar{x}_i \rceil \leq \|\lceil \bar{x}_i \rceil\|_1$ and $\|\lceil \bar{x}_i \rceil\|_1 \leq 2\alpha(1/\delta+1) + L + 1$, where$L = \sum_{1 \leq i \leq \ell} y'_{a_i}$. The change in step 10 is bounded by $\|d\|_1 \leq 2\alpha(1/\delta+2) + m + 1$. Therefore the total change between y' and y'' is bounded by $\mathcal{O}(\frac{m+\alpha}{\delta})$.
Number of components: According to Theorem 7, the number of nonzero components of y'' is equal to the number of non-zero components of x'' which equals $K - \ell + m + 1$ (the number of non-zero components of $\hat{x}$ is bounded by $m+1$). We distinguish between the two cases where $\ell \geq m+1$ and $\ell < m+1$. In the case where $\ell \geq m+1$ the number of components of x'' is smaller than K and hence bounded by C. Consider the case where $\ell < m+1$. By definition of ℓ we know that $L + y'_{\ell+1} \geq (m+2)(1/\delta+2)$. Using the argument in the proof of Theorem 7, we obtain the following inequality:

$$\begin{aligned}\|y'\|_1 &= \sum_{i=\ell+2}^{k} y'_i + y'_{\ell+1} + L = (K - \ell - 1)(1/\delta+2) + y'_{\ell+1} + L \\ &\geq (K - \ell - 1)(1/\delta+2) + (m+2)(1/\delta+2) = (K - \ell + m + 1)(1/\delta+2)\end{aligned}$$

Using that $\|y'\|_1 \leq LIN + 2C$ yields $LIN + 2C \geq (K - \ell + m + 1)(1/\delta+2)$. As $\frac{LIN+2C}{(1/\delta+2)} = \frac{\delta LIN + 2\delta C}{(1+2\delta)} \overset{C \geq \delta LIN}{\leq} \frac{C+2\delta C}{(1+2\delta)} = C$ we obtain that $(K - \ell + m + 1) \leq \frac{LIN+2C}{(1/\delta+2)} \leq C$.
Approximation: According to Theorem 7 we distinguish between the two cases where $\|d\|_1 = 2\alpha(1/\delta+2) + m + 1$ and $\|d\|_1 < 2\alpha(1/\delta+2) + m + 1$. In the second case where $\|d\|_1 < 2\alpha(1/\delta+2) + m + 1$ we know that $\|y''\|_1$ is bounded by $\|x''\|_1$ plus the number of non-zero components of x'' since d is chosen maximally. Hence $\|y''\|_1 \leq \|x''\|_1 + C$. In the case where $\|d\|_1 = 2\alpha(1/\delta+2) + m + 1$, we know $\|y''\|_1 \leq \|\hat{y}\|_1 - 2\alpha(1/\delta+2) - m - 1$. As $\|\hat{y}\|_1 \leq \|\bar{y}\|_1 + \|\lceil \bar{x} \rceil\|_1 \leq \|y'\|_1 + 2\alpha(1/\delta+1) + m + 1$ we get $\|y''\|_1 \leq \|y'\|_1 - \alpha$. Note that we can also make the general claim for y'' that $\|y''\|_1 \leq LIN + 2C - \alpha$. □

The following corollary is an analog to Corollary 2.8 which states what Algorithm 2.13 is doing if the approximation ratio of x' is worse than $(1+\delta)$. We will need this corollary in the next section as we have no true control about the approximation ratio of x'. During the bin packing algorithm new columns might appear in the LP, which might change the optimal solution and therefore the approximation ratio of a solution x'.

Corollary 2.15. *Let $\|x'\|_1 = (1+\delta')LIN$ for some $\delta' \geq \delta$ and $\|x'\|_1 \geq 2\alpha(1/\delta+1)$ and let $\|y'\|_1 \leq LIN + 2C$ for some $C \geq \delta' LIN$ and $\|y'\|_1 \geq (m+2)(1/\delta+2)$. Solutions x' and y' have the same number of non-zero components and for each component we have $x'_i \leq y'_i$. The number of non-zero components of x' and y' is K with $K \leq C$. Then Algorithm 2.13 returns a fractional solution x'' with $\|x''\|_1 \leq \|x'\|_1 - \alpha = (1+\delta')LIN - \alpha$*

and an integral solution y'' where one of the two properties holds: $\|y''\|_1 = \|y'\|_1 - \alpha$ or $\|y''\|_1 = \|x'\|_1 - \alpha + C$. Both x'' and y'' have at most C non-zero components and the distance between y'' and y' is bounded by $\|y'' - y'\|_1 = \mathcal{O}(\frac{m+\alpha}{\delta})$.

Proof. Note that steps 1-3 are basically identical to Algorithm 2.6. Hence Algorithm 2.13 returns by Corollary 6 a fractional solution x'' with $\|x''\|_1 \leq \|x'\|_1 - \alpha$. The distance between the integral solutions y' and y'' are independent of the approximation ratio of x'. Hence the distance between y' and y'' is according to Theorem 2.14 bounded by $\mathcal{O}(\frac{m+\alpha}{\delta})$. The number of non-zero components of x'' and y'' is by the proof of Theorem 2.14 bounded by the number $K \leq C$ of non-zero components of y' or by $\frac{LIN+2C}{1/\delta+2} \leq C$. The approximation guarantee for y'', that $\|y''\|_1 \leq \|y'\|_1 - \alpha$ follows if $\|d\|_1 = 2\alpha(1/\delta+2)+m+1$. If $\|d\|_1 < 2\alpha(1/\delta+2)+m+1$ then $\|y''\|_1 \leq \|x''\|_1 + C \leq \|x'\|_1 + C - \alpha$. We can also make the general claim for y'' that $\|y''\|_1 \leq \|y'\|_1 - \alpha$. □

2.4 AFPTAS for robust bin packing

The goal of this section is to give a fully robust AFPTAS for the bin packing problem using the LP/ILP methods developed in the previous section. For that purpose, we show at first the common way how one can formulate a rounded instance of bin packing as an ILP. In Section 2.4.2, we present a rounding of the item sizes. Therefore, we define abstract properties that need to be fulfilled by the rounding to obtain a good approximation guarantee. In Section 2.4.3, we define operations that modify the rounding as new items arrive over time. The presented dynamic rounding algorithm uses these operations to obtain a feasible rounding that fulfills the proposed properties at every time step of the algorithm. The crucial part is finally the analysis of the dynamic rounding in combination with LP/ILP techniques. Since the ILP and its optimal value are in constant change due to the dynamic rounding, it is difficult to give a bound for the approximation. We develop techniques on how the approximation ratio can be analyzed and bounded.

The *online bin packing problem* is defined as follows: Let $I_t = \{i_1, \dots i_t\}$ be an instance with t items at time step $t \in \mathbb{N}$ and let $s : I_t \to (0, 1]$ be a mapping that defines the sizes of the items. Our objective is to find a function $B_t : \{i_1, \dots, i_t\} \to \mathbb{N}^+$, such that $\sum_{i:B_t(i)=j} s(i) \leq 1$ for all j and minimal $\max_i \{B_t(i)\}$ (i.e. B_t describes a packing of the items into a minimum number of bins). We allow to move few items when creating a new solution B_{t+1} for instance $I_{t+1} = I_t \cup \{i_{t+1}\}$. Sanders et al. [SSS09] and also Epstein and Levin [EL09] defined the *migration factor* to give a measure for the amount of repacking. The migration factor is defined as the total size of all items that are moved between the solutions divided by the size of the arriving item. Formally the migration factor of two packings B_t and B_{t+1} is defined by $\sum_{j \leq t: B_t(i_j) \neq B_{t+1}(i_j)} s(i_j)/s(i_{t+1})$.

2.4.1 LP-Formulation

Let I be an instance of bin packing with M different item sizes $s_1, \dots, s_M$. Suppose that for each item $i_k \in I$ there is a size s_j with $s(i_k) = s_j$. A configuration C_i is a multiset of sizes $\{a(C_i, 1) : s_1, a(C_i, 2) : s_2, \dots a(C_i, m) : s_M\}$ with $\sum_{1 \leq j \leq M} a(C_i, j)s_j \leq 1$, where $a(C_i, j)$ denotes how often size s_j appears in configuration C_i. We denote by C the set of all configurations. Let $|C| = N$. By d_j we denote how many items of size s_j have to be

covered by the LP. We consider the following LP relaxation of the bin packing problem:

$$\begin{aligned} &\min \|x\|_1 \\ &\sum_{C_i \in C} x_i a(C_i, j) \geq d_j \qquad \forall 1 \leq j \leq M \\ &x_i \geq 0 \qquad \forall 1 \leq i \leq N \end{aligned}$$

This LP-formulation was first described by Eisemann [Eis57]. The resulting ILP has n variables which equals the number of configurations N and the number of inequalities m is equal to the number of different item sizes M. Suppose that each size s_j is larger or equal than $\epsilon/2$. Since the number of different item sizes is m, the number of feasible packings for a bin is bounded by $|C| = n \leq (\frac{2}{\epsilon} + 1)^m$. Obviously an optimal integral solution of the LP gives a solution to the bin packing instance I. We denote by $OPT(I)$ the value of an optimal solution. An optimal fractional solution is a lower bound for the optimal value. We denote the optimal fractional solution by $LIN(I)$.

2.4.2 Rounding

We use a rounding technique based on the offline APTAS by Fernandez de La Vega & Lueker [FL81]. As we plan to modify the rounding through the dynamic rounding algorithm we give a more abstract approach on how we can round the items to obtain an approximate packing. At first we divide the set of items into *small* ones and *large* ones. An item i is called *small* if $s(i) < \epsilon/2$, otherwise it is called *large*. Instance I is partitioned accordingly into the large items I_L and the small items I_S. We treat small items and large items differently. Small items can be packed using a greedy algorithm and large items need to be rounded using a rounding function. We define a *rounding function* as a function $R : I_L \mapsto \mathbb{N}$ which maps each large item i to a *group* j. By R^j we denote the set of items being mapped to the same group j, i.e. $R^j = \{i \in I_L \mid R(i) = j\}$. By λ_j^R we denote an item i with $s(i) = \max\{s(i_k) \mid i_k \in R^j\}$. Given an instance I and a rounding function R, we define the rounded instance I^R by rounding the size of every large item $i \in R^j$ for $j \geq 1$ up to the size $s(\lambda_j^R)$ of the largest item in its group. Items in R^0 are excluded from instance I^R. We write $s_R(i)$ for the rounded size of item i in I^R. We define the following properties for a rounding function R.

(A) There is a constant $c \in \mathbb{Q}_{>0}$ such that $\max\{R(i) \mid i \in I_L\} = c/\epsilon^2$,

(B) $|R^i| = |R^j|$ for all $i, j \geq 1$,

(C) there is a constant $d \in \mathbb{Q}_{\geq 1}$ such that $|R^0| = d|R^1|$,

(D) $s(i) \leq s(j) \Leftrightarrow R(i) \geq R(j)$.

Any rounding function fulfilling property (A) has at most $\Theta(1/\epsilon^2)$ different item sizes and hence instance I^R can now be solved approximately using the LP relaxation. The resulting LP relaxation has $\Theta(1/\epsilon^2)$ rows and can be solved approximately with accuracy $(1 + \delta)$ using the max-min resource sharing algorithm [Gri+01] in polynomial time. Based on the fractional solution we obtain an integral solution y of the LP with $\|y\|_1 \leq (1 + \delta)LIN(I^R) + C$ for some additive term $C \geq 0$. We say a packing B *corresponds* to a rounding R and solution y if items in $R^1, \ldots, R^m$ are packed by B according to the integral solution y of the LP. The LP is defined by instance I^R. Items in R^0 are each packed in separate bins.

Lemma 2.16. *Given instance I with items greater than $\epsilon/2$ and a rounding function R fulfilling properties (A) to (D), then $OPT(I^R) \leq OPT(I)$ and $|R^0| \leq \frac{2d}{c}\epsilon OPT(I)$. Let y be an integral solution of the LP for instance I^R with $\|y\|_1 \leq (1+\delta)LIN(I^R)+C$ for some value $C \geq 0$, let B be a packing of I which corresponds to R and y and let $\epsilon' = \frac{2d}{c}\epsilon$. Then*

$$\max_i \{B(i)\} = \|y\|_1 + |R^0| \leq (1+\epsilon'+\delta)OPT(I) + C.$$

Proof. Let $m = \max\{R(i) \mid i \in I_L\}$. Let R^i be the set of items in rounding group i, which corresponds to their rounded sizes and let $\mathcal{R}^i$ be the set of items in R^i, which corresponds to their actual size. Instance I^R contains every item from $R^1, \ldots, R^m$, while items from R^0 are excluded. By property (D) we know, that items in R^i are larger or equal than items in R^{i+1}. By property (C) we find for every item in $\mathcal{R}^1$ a unique item in R^0 with larger or equal size, since the largest item in R^1 to which all items are being rounded up is smaller than any item in R^0. Using property (B) for each item in $\mathcal{R}^{i+1}$ we find a unique larger item in R^i. Therefore we have for every item in the rounded instance I^R an item with larger size in instance I and hence

$$OPT(I^R) \leq OPT(I).$$

Since the packing B corresponds to a solution y, B gives a solution with $\max_i\{B(i)\} \leq (1+\delta)LIN(I^R) + C + |R^0|$ bins and since $LIN(I^R) \leq OPT(I^R) \leq OPT(I)$ we obtain that $\max_i\{B(i)\} \leq (1+\delta)OPT(I) + C + |R^0|$. Further, we can bound $|R^0|$. Since every item in I is of size at least $\epsilon/2$ there is a lower bound for the optimum: $OPT(I) \geq \epsilon/2 \sum_{0\leq i\leq m}|R^i| \geq \epsilon/2 \sum_{0\leq i\leq m}|R^0|/d = \frac{\epsilon(m+1)|R^0|}{2d} \geq \frac{c|R^0|}{2d\epsilon}$. Resolving this inequality, we get $|R^0| \leq \frac{2\epsilon d}{c}OPT(I)$ and hence $|R^0| \leq \epsilon' OPT(I)$. Since c and d are constant we know $|R^0| \leq \epsilon' OPT(I)$ which implies together with the inequality $\max_i\{B(i)\} \leq (1+\delta)OPT(I) + C + |R^0| \leq (1+\epsilon'+\delta)OPT(I) + C$. □

How can we handle the small items? Actually, small items do not make problems at all. We can pack them via FirstFit [CGJ84] into the remaining space of the bins where large items are packed. FirstFit is a greedy algorithm which simply places the current item into the first bin having enough space. A new bin is opened if the item does not fit into any used bin.

Lemma 2.17. *[FL81] Let I be an instance with small and large items and given a packing B of the large items with $\max_i\{B(i)\} \leq K$ for some $K \geq 1$. Packing the small items via FirstFit on top of packing B gives a new packing of instance I which uses*

$$\max\{K, (1+\epsilon)OPT(I) + 1\}$$

bins.

Given instance $I = \{i_1, \ldots, i_t\}$, we define m by $m = \lceil 1/\epsilon^2 \rceil$ if $\lceil 1/\epsilon^2 \rceil$ is even and otherwise $m = \lceil 1/\epsilon^2 \rceil + 1$. By definition m is always even. For every instance I we find a rounding function R with rounding groups $R^0, R^1, \ldots R^m$ which fulfills properties (A)-(D) such that $|R^0| < 2|R^1|$ and $|R^0| \geq |R^1|$.

Algorithm 2.18.

1. *Partition the large items according to the rounding function R in groups $R^0, \ldots, R^m$.*
2. *Round up the size of each large item $i \in R^1, \ldots, R^m$ to $s(\lambda_i^R)$ to obtain instance I^R.*

3. *Compute a fractional solution x of the LP defined by I^R approximately with ratio $(1+\bar{\delta})$.*

4. *Round up each component of the fractional solution to obtain an integral solution y for the LP for instance I^R.*

5. *Pack items in $R^1, \ldots, R^m$ according to the integral solution y.*

6. *Open a bin for each item i with $R(i) = 0$.*

7. *Pack the small items in I_S via FirstFit.*

A solution x of I^R with ratio $(1+\bar{\delta})$ having $m+1$ non-zero components can be computed using max-min resource sharing [Gri+01]. According to Lemma 2.16 and Lemma 2.17, the algorithm described above produces a solution with approximation $\leq (1+\epsilon'+\bar{\delta})OPT + m + 1$ with $\epsilon' \leq \frac{2d}{c}\epsilon \leq 4\epsilon$.

2.4.3 Online Bin Packing

Let us consider the case where items arrive in an online fashion. As a new item arrives, the current rounding has to be adapted to the new instance $I_t \cup \{i_{t+1}\}$. We present operations *insert*, *create*, and *union* that modify the rounding R_t. Based upon these operations we present an algorithm that maintains a structure similar to the properties (A)–(D) of the offline algorithm. Actually we prove that the obtained rounding can be embedded into a rounding $\bar{R}_t$ with properties (A)–(D). The used operations (insert, create and union) worsen the approximation guarantee. We apply the results from the previous section to maintain a constant approximation ratio that depends on ϵ. The presented rounding technique is similar to the one used in [EL09]. In our algorithm we use approximate solutions of ILPs in contrast to the APTAS of Epstein & Levin who solve the ILPs optimally. Handling with approximate ILPs results in a different analysis of the algorithm because many helpful properties of optimal solution are lost.

Note that in an online scenario of bin packing where large and small items arrive online, small items do not need to be considered. We use the same techniques as in [EL09] to pack small items. As a small item arrives we place it via FirstFit [CGJ84]. In this case FirstFit increases the number of bins being used by at most 1 ([FL81]) and the migration factor is zero as we repack no item. Whenever a new large item arrives several small items might also need to be replaced. Every small item in a bin that is repacked by the algorithm, is replaced via FirstFit. Packing small items with this strategy does not increase the number of bins that need to be repacked as a large item arrives. Later on the migration factor will solely be determined by the number of bins that are being repacked. More precisely, we will prove that the number of bins, that need to be repacked is bounded by $\mathcal{O}(1/\epsilon^3)$. Therefore we assume without loss of generality that every arriving item is large, i.e. has a size $\geq \epsilon/2$ (see also [EL09]).

Maintaining properties (A)-(D) for the rounding is not easy online. Every time a new large items arrives, an insertion operation is performed. In the insertion operations, the arriving item is sorted into its corresponding group called R^j and then the largest item of every following group is shifted to the next one (see Figure 1). This way, all items remain sorted (property (D)) and every group except for R^0 has the same cardinality as before. However, simply applying the insertion operation is not enough. As more and more large items arrive, R^0 would grow arbitrarily large and therefore violate property (C). Hence

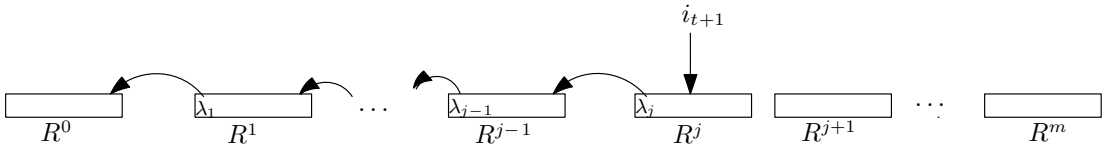


Figure 2.1: Insert operation

we need to find a way to equally distribute items from R^0 to the other groups. That is what the other operations create and union are for. The insert operation is followed by a create or a union operation, depending on the phase the algorithm is in. In the creation phase, items are successively shifted out of R^0 (see Figure 2). At the end of the creation phase two new rounding groups R^1 and R^2 are created from R^0. To make up for the additionally created groups, we have the union phase. In the union phase, items of two rounding groups R^j and R^{j-2} are shifted to rounding groups R^{j-1} and R^{j-3} (see Figure 3). At the end of an union phase, the four rounding groups are merged into two rounding groups with double cardinality and hence the total number of groups is reduced by 2. Using this strategy we are able to maintain properties (A)-(D) when the iteration index is a power of 2. However, in intermediate iterations we can only maintain these properties approximately.

Let $I = \{i_1, \ldots, i_t\}$ be the existing instance as defined above, let R be the corresponding rounding function, let x be a fractional solution of the LP generated for the rounded instance I^R and let B be the current packing of items in I. We define two subgroups of R^0 denoted by $R^{1.5}$ and $R^{2.5}$ in the creation phase, which are also being modified by the operations. Let $I' = I \cup \{i_{t+1}\}$ be the new instance. We use the following operations that modify the current rounding R, the packing B and the fractional and integral LP solution x and y. We denote with R', B', x' and y' the new rounding, packing and fractional/integral LP solutions for instance I'

Insertion Step

Find the largest j with $s(\lambda_j^R) \geq s(i_{t+1})$. Set $R'(i_{t+1}) = j$ and $B'(i_{t+1}) = B(\lambda_j^R)$. For every $k = 1, \ldots, j$ we define $R'(\lambda_k^R) = k - 1$ and $B'(\lambda_k^R) = B(\lambda_{k-1}^R)$. Set $x' = x$ and $y' = y$.

Modified Insertion Step

During the creation phase, the algorithm uses the modified insertion operation. Find the largest j ($j = 1.5$ and $j = 2.5$ included) with $s(\lambda_j^R) \geq s(i_{t+1})$. Set $R'(i_{t+1}) = j$ and $B'(i_{t+1}) = B(\lambda_j^R)$. For every $k = 1, 4, 5, \ldots, j$ we define $R'(\lambda_k^R) = k - 1$ and $B'(\lambda_k^R) = B(\lambda_{k-1}^R)$. For every $k = 1.5, 2, 2.5, 3$ we define $R'(\lambda_k^R) = k - 0.5$ and $B'(\lambda_k^R) = B(\lambda_{k-0.5}^R)$. Set $x' = x$ and $y' = y$.

Creation Phase

The creation phase consists of k creation steps, where $k = |R^1|$. At the end of each creation phase we intend to have new rounding groups R^1 and R^2 created from the subgroups of

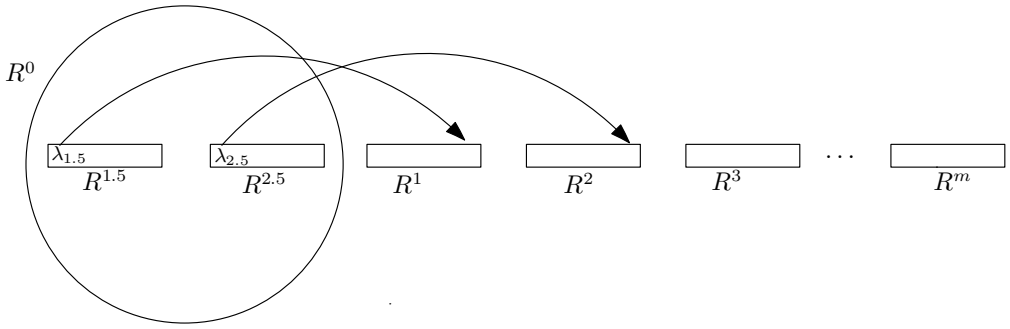


Figure 2.2: Create operation

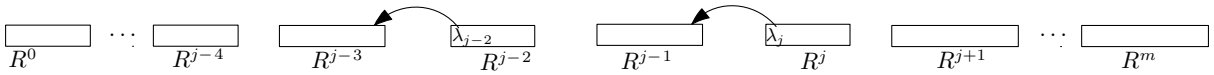


Figure 2.3: Union operation

R^0 named $R^{1.5}$ and $R^{2.5}$. At the beginning of the creation phase we always have $|R^0| = 2k$ and $R^{1.5}$ and $R^{2.5}$ are empty. In the first step we change the rounding group for all items i with $R(i) = j \geq 1$ to $R'(i) = j + 2$. Furthermore we say the k largest items of R^0 belong to $R^{1.5}$ and the k smallest items belong to $R^{2.5}$. In each of the k creation steps we change the rounding function for the largest items $\lambda^R_{1.5}$ and $\lambda^R_{2.5}$. Set $R'(\lambda^R_{1.5}) = 1$ and $R'(\lambda^R_{2.5}) = 2$. Since items $\lambda^R_{1.5}$ and $\lambda^R_{2.5}$ are moved from R^0 to R^1 and R^2 they have to be covered by the LP. Therefore we increase the value of the LP solution by $x'_i = x_i + 1$, $x'_j = x_j + 1$ and $y'_i = y_i + 1$, $y'_j = y_j + 1$, where i, j are defined such that $C_i = \{1 : s_{R'}(\lambda^R_{1.5})\}$ and $C_j = \{1 : s_{R'}(\lambda^R_{2.5})\}$. For $k \neq i, j$ set $x'_k = x_k$ and $y'_k = y_k$.

Union Phase

The union phase consists of k union steps, where $k = |R^1|$. At the end of each union phase four roundings groups are merged into two rounding groups with double cardinality. For the first union step we determine the largest index j with $|R^j| < |R^{j+1}|$. If there is no such index then set $j = m$. In each step now set $R'(\lambda^R_j) = j - 1$ and $R'(\lambda^R_{j-2}) = j - 3$ and for the other items i we define $R'(i) = R(i)$. Modify the packing for λ^R_j and λ^R_{j-2} by $B'(\lambda^R_j) = B(\lambda^R_{j-2})$ and place λ^R_{j-2} into a new bin. Modifying the packing this way implies that we have to change one configuration of the fractional and integral LP solution x and y and add one configuration for the additional bin. Let C_i be the configuration used by $B'(\lambda_j)$. Configuration C_i is replaced by a configuration $\hat{C}_i$ where an item of size $s_{R'}(\lambda^R_{j-2})$ is exchanged by an item of size $s_{R'}(\lambda^R_j)$. Furthermore we add another configuration C_ℓ with an item of size $s_{R'}(\lambda^R_{j-2})$.

Note that each repacking that we perform in the operations is valid because we always replace items by smaller ones.

The new packing B' is created in a way that it corresponds to the new integer solution y'. We have to prove that the solution y' is feasible. Note also that in a creation operation and in a union operation two additional non-zero components of size 1 might be created.

Lemma 2.19. *Applying any operation above on a rounding R and ILP solution y with corresponding packing B defines a new rounding R' and a new integral solution y'. Solution y' is a feasible solution of the LP for instance $I^{R'}$.*

Proof. We have to analyze how the LP for instance $I^{R'}$ changes in comparison to the LP for instance I^R.
Insertion Operation: The right hand side of the LP derived from R' does not change at all since the right hand side is determined by the cardinalities $|R'^1| = |R^1|, \ldots, |R'^m| = |R^m|$. For some $j \geq 1$ let R^j be the rounding group where the new item is inserted. By construction of the insertion operation for each rounding group R^ℓ with $\ell = 1, \ldots, j$, there is one item that is inserted into group R'^ℓ and one item that is shifted out. Let ι_ℓ^R be the second largest item of rounding group R^ℓ. Since the largest item λ_ℓ^R in group R^ℓ is shifted to the next group, the size $s_{R'}(i)$ of item i in a group R^ℓ is defined by $s_{R'}(i) = \iota_\ell^R$. Therefore each item in $I^{R'}$ is rounded to the previous smaller value since $s(\iota_\ell^R) \leq s(\lambda_\ell^R)$. Hence configurations of the LP solution for I^R can be transformed into feasible configurations for $I^{R'}$ i.e. $\|y'\|_1 = \|y\|_1$.
Creation Operation: Note that the rounding groups R^ℓ, for $\ell = 1, \ldots, m+2$ remain identical; i.e. $R'^\ell = R^\ell$. The groups R'^1 and R'^2 get both a new item, but of smaller size. Therefore the sizes $s_r(i)$ of all items $i \in I^R$ are not modified by a creation operation. We have $s_R(i) = s_{R'}(i)$ for items in groups $R^1, \ldots, R^{m+2}$. Therefore the matrix $A = (a((i,j)))$ remains the same. Only the right hand side b' of the LP from instance $I^{R'}$ is modified (i.e. $\|b' - b\|_1 = 2$). As two new configurations are being added to x and y they cover exactly the enhanced right hand side and are therefore a feasible solution of the LP from instance $I^{R'}$.
Union Operation: In the union operation we basically change only 4 rounding groups. Suppose we merge rounding group R^{j-3} with R^{j-2} and rounding group R^{j-1} with R^j. While the size of $|R'^{j-3}| = |R^{j-3}| + 1$ and $|R'^{j-1}| = |R^{j-1}| + 1$ is incremented the size of $|R'^j| = |R^j| - 1$ and $|R'^{j-2}| = |R^{j-2}| - 1$ is reduced. Similar to the creation operation, this leads to a change in the right hand side of the LP. Two components of the right hand side, which correspond to $s(\lambda_j^{R'})$ and $s(\lambda_{j-2}^{R'})$ are reduced by 1 and two other components, which correspond to the $s(\lambda_{j-1}^{R'})$ and $s(\lambda_{j-3}^{R'})$ are increased by 1. Furthermore the sizes of items in R'^j and R'^{j-2} are equal or smaller than the sizes of items in R^j and R^{j-2} since $s_{R'}(i) = \iota_j^R$ for all items $i \in R'^j$ and $s_{R'}(i) = \iota_{j-2}^R$ for all items $i \in R'^{j-2}$. λ_j^R and λ_{j-2}^R are shifted to the next rounding groups. Consider a feasible configuration C of the LP for instance I^R. Then the modified configuration $\bar{C}$ (with replaced item sizes) is also feasible in the LP for instance $I^{R'}$. The new solutions x' and y' use the modified configurations and cover the right hand side of the LP. □

The operations are used as described in Algorithm 2.20 below. We apply the algorithm on a rounding function R_0 and instance $I_0 = \{i_1, \ldots i_T\}$. We suppose that $|R_0^0| = |R_0^1| = \ldots = |R_0^m| = K$ for some $K > 0$ and hence $T = K(m+1)$. An improve$(a, x, y, \bar{\delta})$ statement stands for a call of Algorithm 2.13 with improvement $\alpha = a$, fractional solution x and integral solution y. The variable part x^{var} is defined by $x^{var} = 2\frac{\alpha(1/\bar{\delta}+1)}{\|x\|_1}x$. After an improve call the packing is changed according to the new integral solution. Since during a creation operation and a union operation two additional non-zero components of size 1 might appear, we change the parameter ℓ of Algorithm 2.13 slightly to ℓ'. Parameter ℓ' is defined maximally such that the sum of the smallest components $y'_1, \ldots, y'_\ell$ are $\sum_{1 \leq i \leq \ell} y'_{a_i} \leq (m+2)(1/\delta + 2) + 2$. The two additional non-zero components belong to components $y_1, \ldots y_{\ell+2}$ and are therefore reduced in step 6 along with the others.

Figure 2.4: Using Algorithm 2.20 on a rounding with $m = 6$

phases	$\lvert R^0\rvert$	$\lvert R^1\rvert$	$\lvert R^2\rvert$	$\lvert R^3\rvert$	$\lvert R^4\rvert$	$\lvert R^5\rvert$	$\lvert R^6\rvert$	$\lvert R^7\rvert$	$\lvert R^8\rvert$
start	K	K	K	K	K	K	K	0	0
insertion	$2K$	K	K	K	K	K	K	0	0
creation	K	K	K	K	K	K	K	K	K
union	$2K$	K	K	K	K	$2K$	$2K$	0	0
creation	K	K	K	K	K	K	K	$2K$	$2K$
union	$2K$	K	K	$2K$	$2K$	$2K$	$2K$	0	0
creation	K	K	K	K	K	$2K$	$2K$	$2K$	$2K$
union	$2K$	$2K$	$2K$	$2K$	$2K$	$2K$	$2K$	0	0

Algorithm 2.20.

```
for i := 1 to K do
    get new item;
    improve(1, x, y, δ̄); insert;
for i := 1 to m/2 do
    /* Creation Phase                                   */
    for j := 1 to K do
        get new item;
        improve(1, x, y, δ̄);
        modified insert;
        create;
    /* Union Phase                                      */
    for j := 1 to K do
        get new item;
        improve(2, x, y, δ̄);
        insert;
        union;
```

In Figure 4 we present how the algorithm changes the rounding groups for $m = 6$. The table presents the state of each rounding groups after each phase. One can see that after the execution of Algorithm 2.20 each rounding group has exactly $2K$ items and the number of rounding groups is the same as before. Since the number of groups has to be constant (property (A)), this is exactly what we want. We prove the general case for arbitrary m: Every rounding has exactly $2K$ items after the execution of Algorithm 2.20.

Lemma 2.21. *Let R_0 be the rounding function at the beginning of the algorithm. Suppose that every rounding group $R_0^0, \ldots, R_0^m$ has exactly K items. Then after the execution of the algorithm above, the computed rounding function R_T after $T = K(m+1)$ insertions has $m+1$ rounding groups $R_T^0, \ldots, R_T^m$ with $\lvert R_T^\ell\rvert = 2K$ for $\ell = 0, \ldots, m$.*

Proof. The algorithm starts with a rounding function that contains exactly T items. After the first K insertion steps rounding function R_K is of the form: $\lvert R_K^0\rvert = 2K, \lvert R_K^1\rvert = K, \ldots, \lvert R_K^m\rvert = K$ since K items are shifted to R^0 while the cardinalities of the other rounding groups remain the same. During the next K arrivals, the algorithm is in the creation phase. We perform a creation operation after each insertion. For each item shifted

to R^0, two items are shifted to the new created groups R^1 and R^2. At the end of the first creation phase, the rounding function R_{2K} satisfies $|R^0_{2K}| = K, |R^1_{2K}| = K, \ldots, |R^{m+2}_{2K}| = K$. In the following union phase, the rounding groups R^{m+2}, R^{m+1} and R^m, R^{m-1} are merged together. For each union operation, one item is shifted from R^{m+2} to R^{m+1} and another from R^m to R^{m-1}. Since there are K insert operations in the union phase, rounding group $|R^0_{3K}| = 2K, |R^1_{3K}| = K, \ldots, |R^{m-2}_K| = K, |R^{m-1}_K| = 2K, |R^m_K| = 2K$. After the next creation and union phase, the number of rounding groups is also $m+1$. On the other hand we have two additional groups $|R^{m-3}_{5K}| = |R^{m-2}_{5K}| = 2K$. After $j < m/2$ creation and union phases the rounding function R_{2jK+K} is by induction of the form $|R^0_{2jK+K}| = 2K, |R^1_{2jK+K}| = K, \ldots, |R^{m-2j}_{2jK+K}| = K, |R^{m-2j+1}_{2jK+K}| = 2K, \ldots, |R^m_{2jK+K}| = 2K$. This can be proved by induction on j. For $j = m/2-1$ we get $|R^0_{2mK-K}| = 2K, |R^1_{2mK-K}| = K, |R^2_{2mK-K}| = K, |R^3_{2mK-K}| = 2K, \ldots |R^m_{2mK-K}| = 2K$. Therefore, after one additional creation and union phase, we obtain $m+1$ groups $|R^\ell_{2mK+K}| = 2K$ for $\ell = 0, \ldots, m+1$. $\square$

The following algorithm is our final online AFPTAS for the classical bin packing problem. Let S_t be the sum of all item sizes $s(i_1) + \ldots + s(i_t)$.

Algorithm 2.22.

- *While $S_t \leq (m+2)(1/\bar{\delta} + 4)$ and $(m+1)$ does not divide t get the new item i_{t+1} and use the offline Algorithm 2.18 with an LP of approximation ratio $(1+\bar{\delta})$.*
- *Afterwards use Algorithm 2.20 repetitively to obtain a packing for each instance*

By using the offline AFPTAS for small instances, we can make sure that Algorithm 2.20 is started with a suitable rounding function. Since Algorithm 2.20 always produces a rounding function fulfilling properties (A) to (D) and $m+1$ divides the current number of items t, every rounding group $R^0, \ldots, R^m$ has the same number of item sizes as the algorithm leaves the while-loop in the first step.

In the following we give a bound for the rounding functions R_t that we produce in every step of the algorithm. It remains to prove that the approximation during the execution of Algorithm 2.20 can be bounded. Therefore we define a relation between rounding functions. Let R and $\bar{R}$ be two rounding functions, with $\bar{R}$ having $\bar{m}$ rounding groups for some $\bar{m} \in \mathcal{O}(1/\epsilon^2)$. We can *embed* R into $\bar{R}$ in symbols $R \leq \bar{R}$, if $|R^0| \leq |\bar{R}^0|$ and for every item $i \in I \setminus \bar{R}^0$ we have $s_R(i) \leq s_{\bar{R}}(i)$. A relation $R \leq \bar{R}$ always implies that $|R^0| + OPT(I^R) \leq |\bar{R}^0| + OPT(I^{\bar{R}})$.

Lemma 2.23. *For each $t \in \mathbb{N}^+$, we can embed R_t into a function $\bar{R}_t$, which fulfills properties (A) to (D). Rounding function $\bar{R}_t$ has parameter $c \geq 1/4$ for property (A) and $d \leq 2$ for property (C).*

Proof. Since we basically shift largest items to the following rounding group we designed operations insertion, creation and union in a way that property (D) is never being violated by any R_j for $j \leq t$. As shown in the proof of Lemma 2.19 the number of rounding groups remains constant between $m+1$ at the end of a union phase and $m+3$ during the creation and union phase. Suppose the algorithm above has started with a number of items $T < t \leq 2T$ and rounding groups $R^0_T, \ldots R^m_T$,which are being modified by the algorithm. We define the rounding function $R = \bar{R}_t$ in which R_t can be embeded in the following way: Function $\bar{R}$ has rounding groups $\bar{R}^0, \ldots, \bar{R}^{\lfloor \frac{t}{2K} \rfloor - 1}$ with $|\bar{R}^1| = \ldots = |\bar{R}^{\lfloor \frac{t}{2K} \rfloor - 1}| = 2K$ and $|\bar{R}^0| = 2K + (t \mod 2K)$. Since every rounding group

of $\bar{R}$ except $\bar{R}^0$ has the same number of items, the rounding function $\bar{R}$ fulfills property (B). Rounding function $\bar{R}$ fulfills property (C) because $2K \leq |\bar{R}^0| \leq 4K$ and $|\bar{R}^1| = 2K$. This implies constant $d \leq 2$. We prove property (A) by giving an upper and a lower bound for $\max_i \left\{\bar{R}(i)\right\}$ that are both in $\Theta(\frac{1}{\epsilon^2})$. Recall that $T/K = m+1$. On the one hand we get $\max_i \left\{\bar{R}(i)\right\} = \lfloor \frac{t}{2K} \rfloor - 1 \leq \frac{t}{2K} - 1 \leq \frac{2T}{2K} - 1 \leq (m+1) - 1 = m \leq \lceil \frac{1}{\epsilon^2} \rceil + 1 \leq \frac{1}{\epsilon^2} + 2$. On the other hand $\max_i \left\{\bar{R}(i)\right\} = \lfloor \frac{t}{2K} \rfloor - 1 \geq \lfloor \frac{T}{2K} \rfloor - 1 \geq \lfloor \frac{m+1}{2} \rfloor - 1 \geq \frac{m}{2} - 1 = \frac{1}{2\epsilon^2} - 1$. Since $\epsilon \leq 1/2$ weg get $c \geq 1/4$. It remains to prove that we can embed R_t in $\bar{R}$ i.e. $R_t \leq \bar{R}$. Since R^0 never exceeds $2K$ items and $2K \leq 4K$ we get $|R_t^0| \leq |\bar{R}^0|$. According to the proof of Lemma 2.19 and the construction of the creation operation, R_t is during the creation phase of the following form: $|R_t^0| = 2K - a, |R_t^1| = a, |R_t^2| = a, |R_t^3| = \ldots = |R_t^j| = K, |R_t^{j+1}| = \ldots = |R_t^{m+2}| = 2K$ for some $a \leq K$ and $j \leq m+2$. Rounding function $\bar{R}$ has in every rounding group $\bar{R}^j$ for $j \geq 1$ exactly $2K$ items. Since property (D) holds for both rounding function R_t and $\bar{R}$, the rounding groups $R_t^{j+1}, \ldots, R_t^{m+2}$ contain the same items as the last $m+2-j$ rounding groups of $\bar{R}$. Items in these groups are therefore rounded identically. For some $\bar{m}$ let $\bar{R}^{\bar{m}}$ be the rounding group which contains the same items as R_t^{j+1}. Since rounding groups $R_t^3, \ldots, R_t^j$ each contain exactly K items, the rounding groups $\bar{R}^1, \ldots, \bar{R}^{\bar{m}-1}$ contain the items of exactly two rounding groups. Therefore the items in $\bar{R}^1, \ldots, \bar{R}^{\bar{m}-1}$ are rounded to a smaller size compared to using $\bar{R}$. Items that belong to R_t^1 and R_t^2 are contained in $\bar{R}^0$ and by definition do not need to be considered. Hence, any rounding function R_t which is in an creation phase can be embedded into an $\bar{R}$. By construction of the union operation and the proof of Lemma 2.19, R_t is during the union phase of the form $|R_t^0| = K + a, |R_t^1| = \ldots = |R_t^{j-4}| = K, |R_t^{j-3}| = K + a, |R_t^{j-2}| = K - a, |R_t^{j-1}| = K + a, |R_t^j| = K - a, |R_t^{j+1}| = \ldots = |R_t^{m+2}| = 2K$ for some $a \leq K$ and $j \leq m+2$. As shown in the union phase, items in $R_t^{j+1}, \ldots, R_t^{m+2}$ are rounded equally in $\bar{R}$. As the sum of R_t^{j-1} and R_t^j is $2K$ and the sum of R_t^{j-3} and R_t^{j-2} is $2K$ the items of R_t^{j-1} and R_t^j and the items in R_t^{j-3} and R_t^{j-2} belong in $\bar{R}$ to the same rounding group and are hence rounded equally or to a smaller size compared to using $\bar{R}$. Items in $R_t^1, \ldots, R_t^{j-4}$ are each of size K and are rounded equally or to a smaller size than using $\bar{R}$ since the same argument as in the creation phase holds. □

Define $\bar{\epsilon}$ by $\bar{\epsilon} = \frac{1}{16}\epsilon$. As $\bar{R}_t$ fulfills property (A) to (D), we obtain by Lemma 2.16 and $R_t \leq \bar{R}_t$ the following two equations for every t:

1. $OPT(I^{\bar{R}_t}) \leq OPT(I_t)$
2. $|R_t^0| \leq |\bar{R}_t^0| \leq \frac{2d}{c}\epsilon OPT(I_t) \leq \bar{\epsilon} OPT(I_t)$

Note that since $\epsilon \leq 1/2$ we have $\bar{\epsilon} \leq \frac{1}{32}$. Recall that $R_t \leq \bar{R}_t$ implies that $|R_t^0| + OPT(I^{R_t}) \leq |\bar{R}_t^0| + OPT(I^{\bar{R}_t})$ and that $LIN(I^{R_t}) + m \geq OPT(I^{R_t})$ (rounding up a basic feasible solution). Let us discuss how the methods from the previous section apply to the presented online algorithm. The procedure improve is implemented by using Algorithm 2.13 in order to get an improved solution for instance I^{R_t}. Algorithm 2.13 is applied using $\bar{\delta}$ as the approximation parameter. In the following lemma we prove that applying Algorithm 2.13 to improve a solution for I^{R_t} impacts the overall approximation $\Delta = \bar{\epsilon} + \bar{\delta} + \bar{\epsilon}\bar{\delta}$ in the same way. We define $C = \Delta OPT(I_t) + m$.

Theorem 2.24. *Given a rounding function R_t and an LP defined for I^{R_t}. Let x be a fractional solution of the LP with $\|x\|_1 + |R_t^0| \leq (1+\Delta)OPT(I_t)$ and $\|x\|_1 \geq 2\alpha(1/\bar{\delta}+1)$ and $\|x\|_1 = (1+\delta')LIN(I^{R_t})$ for some $\delta' > 0$. Let y be an integral solution of the LP*

with $\|y\|_1 \geq (m+2)(1/\bar{\delta}+2)$ *and corresponding packing* B_t *such that* $\max_i B_t(i) = \|y\|_1 + |R_t^0| \leq (1+2\Delta)OPT(I_t) + m$. *Suppose* x *and* y *have the same number* $\leq C$ *of non-zero components and for all components* i *we have* $y_i \geq x_i$. *Then using Algorithm 2.13 on* x *and* y *returns new solutions* x' *with* $\|x'\|_1 + |R_t^0| \leq (1+\Delta)OPT(I_t) - \alpha$ *and integral solution* y' *with corresponding packing* B'_t *such that*

$$\max_i B'_t(i) \leq (1+2\Delta)OPT(I_t) + m - \alpha.$$

Further, both solutions x' *and* y' *have the same number* $\leq C$ *of non-zero components and for each component we have* $x'_i \leq y'_i$.

Proof. As shown in the following, Algorithm 2.13 maintains the property that x and y have the same number of non-zero components and that $x_i \leq y_i$ since we can use Theorem 2.14 and Corollary 2.15. By condition we have $\max_i B_t(i) = \|y\|_1 + |R_t^0| \leq (1+2\Delta)OPT(I_t) + m$. Since $OPT(I_t) \leq OPT(I_t^R) + |R_t^0|$ we obtain for the integral solution y that $\|y\|_1 \leq 2\Delta OPT(I_t) + m + OPT(I_t^R) \leq 2\Delta OPT(I_t) + m + LIN(I_t^R) + m$. Hence by definition of C we get $\|y\|_1 \leq LIN(I_t^R) + 2C$. This is one requirement to use Theorem 2.14 or Corollary 2.15. We look at the cases separately where on the one hand $\delta' \leq \bar{\delta}$ and on the other hand $\delta' > \bar{\delta}$.

Case 1, $\delta' \leq \bar{\delta}$: At first we give an upper bound for $LIN(I^{R_t})$: We get $LIN(I^{R_t}) \leq OPT(I^{R_t}) \leq OPT(I^{R_t}) + |R_t^0| \leq OPT(I^{\bar{R}_t}) + |\bar{R}_t{}^0| \leq (1+\bar{\epsilon})OPT(I_t)$ using that $R_t \leq \bar{R}_t$. This implies that $\bar{\delta}LIN(I^{R_t}) \leq \bar{\delta}OPT(I^{R_t}) \leq (\bar{\delta} + \bar{\delta}\bar{\epsilon})OPT(I_t) < C$. Algorithm 2.13 returns by Theorem 2.14 a solution x' with $\|x'\|_1 \leq (1+\bar{\delta})LIN(I^{R_t}) - \alpha$ and an integral solution y' with $\|y'\|_1 \leq \|x'\|_1 + C$ or $\|y'\|_1 \leq \|y\|_1 - \alpha$. For the term $\|x'\|_1 + |R_t^0|$ we get $\|x'\|_1 + |R_t^0| \leq (1+\bar{\delta})OPT(I^{R_t}) - \alpha + |R_t^0|$. Using that R_t can be embedded in $\bar{R}_t$ we get $|R_t^0| + OPT(I^{R_t}) \leq |\bar{R}_t{}^0| + OPT(I^{\bar{R}_t}) \leq OPT(I_t) + \bar{\epsilon}OPT(I_t)$. Therefore

$$\begin{aligned}\|x'\|_1 + |R_t^0| &\leq \bar{\delta}OPT(I^{R_t}) - \alpha + OPT(I_t) + \bar{\epsilon}OPT(I_t)\\ &\leq (\bar{\delta} + \bar{\delta}\bar{\epsilon})OPT(I_t) - \alpha + (1+\bar{\epsilon})OPT(I_t) \leq (1+\Delta)OPT(I_t) - \alpha.\end{aligned}$$

In the case where $\|y'\|_1 \leq \|x'\|_1 + C$ we can bound the number of bins of the new packing B' by $\max_i B'_t(i) = \|y'\|_1 + |R_t^0| \leq \|x'\|_1 + |R_t^0| + C \leq (1+\Delta)OPT(I_t) - \alpha + C = (1+2\Delta)OPT(I_t) + m - \alpha$. In the case that $\|y'\|_1 \leq \|y\|_1 - \alpha$ we obtain $\max_i B'_t(i) = \|y'\|_1 + |R_t^0| \leq \|y\|_1 - \alpha + |R_t^0| = \max_i B_t(i) - \alpha \leq (1+2\Delta)OPT(I_t) + m - \alpha$.

Case 2, $\delta' > \bar{\delta}$: By condition we have $\|x\|_1 + |R_t^0| \leq (1+\Delta)OPT(I_t)$. Since $OPT(I_t) \leq OPT(I^{R_t}) + |R_t^0|$ we obtain for the solution x that $\|x\|_1 \leq \Delta OPT(I_t) + OPT(I^{R_t}) \leq \Delta OPT(I_t) + LIN(I^{R_t}) + m$. Hence by definition of C this implies $\|x\|_1 \leq LIN(I^{R_t}) + C$ and therefore $\delta' LIN(I^{R_t}) < C$, which fulfills the requirements of Corollary 2.15. Using Algorithm 2.13 on solutions x with $\|x\|_1 = (1+\delta')LIN(I^{R_t})$ and y with $\|y\|_1 \leq LIN(I^{R_t}) + 2C$ we obtain by Corollary 2.15 a fractional solution x' with $\|x'\|_1 \leq \|x\|_1 - \alpha$ and an integral solution y' with either $\|y'\|_1 \leq \|y\|_1 - \alpha$ or $\|y'\|_1 \leq \|x\|_1 + C - \alpha$. So for the new packing B' we can guarantee, that $\max_i B'_t(i) = \|y'\|_1 + |R_t^0| \leq \|y\|_1 - \alpha + |R_t^0| = \max_i B_t(i) - \alpha \leq (1+2\Delta)OPT(I_t) + m - \alpha$ if $\|y'\|_1 \leq \|y\|_1 - \alpha$. If $\|y'\|_1 \leq \|x\|_1 + C - \alpha$, we can guarantee that $\max_i B'_t(i) = \|y'\|_1 + |R_t^0| \leq \|x\|_1 + |R_t^0| + C - \alpha \leq (1+\Delta)OPT(I_t) + C - \alpha \leq (1+2\Delta)OPT(I_t) + m - \alpha$. Furthermore we know by Corollary 2.15 that x' and y' have at most C non-zero components. □

Set $\bar{\delta} = \bar{\epsilon}$. Then $\Delta = 2\bar{\epsilon} + \bar{\epsilon}^2 = \mathcal{O}(\epsilon)$. We get the central theorem:

Theorem 2.25. *Algorithm 2.22 is a fully robust AFPTAS for the bin packing problem.*

Proof. While instances are small Algorithm 2.22 uses the offline AFPTAS (see Algorithm 2.18). Using Algorithm 2.18, we get a packing B_t for instance I_t that uses at most $\max_i B_t(i) \leq (1+\epsilon'+\bar{\delta})OPT(I_t) + \frac{1}{\epsilon^2} + 1$ bins, where $\epsilon' \leq 4\epsilon < \bar{\epsilon}$. Since the instance is small the migration factor is bounded although we might repack every single item. Let τ be the first index where the algorithm leaves the while-loop. By condition we are in the while loop while $S_t \leq (m+2)(1/\bar{\delta}+4)$ and t does not divide $m+1$. Hence $S_\tau \leq (m+2)(1/\bar{\delta}+4) + m = \mathcal{O}(1/\epsilon^3)$. The migration factor for instances I_t with $t \leq \tau$ is therefore bounded by $\frac{2}{\epsilon}S_t = \mathcal{O}(1/\epsilon^4)$ since every arriving item has size at least $\epsilon/2$. The approximation guarantee for small instances is bounded by $\max_i B_t(i) \leq (1+\bar{\delta}+\bar{\epsilon})OPT(I_t) + m + 1$. In the following we consider large instances I_t with $t \geq \tau$.

Full robustness: The migration factor for some consecutive packings B_t and B_{t+1} is bounded by the migration of the improve-call plus the migration of an insertion and an union operation. The operations create requires no shifting of items at all. As proven in the previous section, an improve-call changes at most $\mathcal{O}(m/\bar{\delta})$ components of a solution y. Since the arriving item is large with size $\geq \epsilon/2$, changing a complete configuration requires migration of at most $\mathcal{O}(1/\epsilon)$. Combined this results in a migration factor for the improve-call $\mathcal{O}(m/\Delta^2) = \mathcal{O}(1/\epsilon^4)$ if we use Algorithm 2.13. By construction of the insertion operation it shifts in worst case one item per rounding group. Having $\mathcal{O}(1/\epsilon^2)$ rounding groups this gives a migration factor of at most $\mathcal{O}(1/\epsilon^3)$. Therefore the complete migration is bounded by $\mathcal{O}(1/\epsilon^4)$.

Running time: The running time is dominated by the max-min resource sharing (see Algorithm 2.13) and the number of non-zero components. The number of non-zero components is bounded by $\Delta OPT(I_t) + m \leq \Delta t + \frac{1}{\epsilon^2} + 1$ and is therefore polynomial in $\frac{1}{\epsilon}$ and t. As the running time for the max-min resource sharing is also polynomial in $\frac{1}{\epsilon}$ (see [Gri+01]), the running time is clearly polynomial in t and $\frac{1}{\epsilon}$.

Approximation: We prove by induction that four properties hold for any packing B_t and corresponding LP solutions. Given fractional solutions x and integral solution y of the LP defined by instance I^{R_t}. Properties (2)-(4) are necessary to apply Theorem 2.25 and property (1) provides the wished approximation ratio for the bin packing problem.

1. packing B_t uses at most $(1+2\Delta)OPT(I_t) + m$ bins
2. $\|x\|_1 + |R_t^0| \leq (1+\Delta)OPT(I_t)$
3. for every configuration i we have $x_i \leq y_i$
4. x and y have the same number of non-zero components and that number is bounded by $\Delta OPT(I_t) + m$

To apply Theorem 2.24 we furthermore need a guaranteed minimal size for $\|x\|_1$ and $\|y\|_1$. According to Theorem 2.24 integral solution y needs $\|y\|_1 \geq (m+2)(1/\bar{\delta}+2)$ and $\|x\|_1 \geq 4(1/\bar{\delta}+1)$ as we set at most $\alpha = 2$. By condition of the while-loop we know that any instance $S_t \geq (m+2)(1/\bar{\delta}+6)$. Since $OPT(I_t) \leq \|y\|_1 + |R_t^0| \leq \|y\|_1 + \bar{\epsilon}OPT(I_t)$ we get $\|y\|_1 \geq (1-\bar{\epsilon})OPT(I_t) = (1-\bar{\delta})OPT(I_t)$. By $OPT(I_t) \geq (m+2)(1/\bar{\delta}+4)$ we finally get that $\|y\|_1 \geq (1-\bar{\delta})(m+2)(1/\bar{\delta}+6) \geq (m+2)(1/\bar{\delta}+6) - (m+2)(1+6\bar{\delta}) \geq (m+2)(1/\bar{\delta}+6) - 4(m+2) = (m+2)(1/\bar{\delta}+2)$. Since $OPT(I_t) \leq \|x\|_1 + m + |R_t^0|$ we obtain by the same argument that $\|x\|_1 \geq (m+2)(1/\bar{\delta}+2) - m \geq (m+2)(1/\bar{\delta}+1)$ and since $m = 1/\epsilon \overset{\epsilon\leq\bar{\delta}}{\geq} 1/\bar{\delta} \geq 2$ we get that $\|x\|_1 \geq 4(1/\bar{\delta}+1)$.

In the case that $t = \tau$ we have by the offline algorithm that the number of non-zero components $= m+1 \leq \Delta OPT(I_t) + m$ since $OPT(I_t) \geq S_t > 1/\Delta$. The number of used

bins is bounded by $\max_i B_t(i) < (1+\bar{\delta}+\bar{\epsilon})OPT(I_t)+m+1 < (1+2\Delta)OPT(I_t)+m$ (note $\epsilon' < \bar{\epsilon}$) and property (2) is fulfilled for the same reason. Furthermore in the offline algorithm every component x_i is rounded up to obtain the integral component y_i. Therefore all properties (1)-(4) are fulfilled for $t \leq \tau$ and the induction basis holds. Now let B_t be a packing for $t > \tau$ for instance I_t with solutions x and y of the LP defined by I^{R_t}. Suppose by induction that property (1)-(4) hold. We have to prove that these properties also hold for B_{t+1} and the corresponding solutions of the LP defined by $I^{R_{t+1}}$. Packing B_{t+1} is created by using an improve call for x and y followed by an insertion operation and optional, an union or a creation operation.

improve: Let x' be the resulting fractional solution of Algorithm 2.13, let y' be the resulting integral solution of Algorithm 2.13 and let B'_t be the corresponding packing. Properties (1)-(4) are fulfilled for x, y and B_t by induction hypothesis. Hence we can use Theorem 2.24. By Theorem 2.24 properties (1)-(4) are then still fulfilled for x', y' and B'_t and moreover we get $\|x'\|_1 + |R_t^0| \leq (1+\Delta)OPT(I_t) - \alpha$ and $\max_i B'_t(i) \leq (1+2\Delta)OPT(I_t) + m - \alpha$ for $\alpha = 2$ or $\alpha = 2$.

operations: First we take a look at how the operations modify $\|x'\|_1$, $\|y'\|_1$ and $|R_t^0|$. By construction of the insertion operation, the LP solutions x' and y' are not modified while $|R_t^0|$ increases by 1. By construction of the creation operation $\|x'\|_1$ and $\|y'\|_1$ are increased by 2 and $|R_t^0|$ decreases by 2. By construction of the union operation, $\|x'\|_1$ and $\|y'\|_1$ are increased by 1 and $|R_t^0|$ remains constant. Property (1): Let x'' be the fractional solution and y'' be the integral solution after using operations on x' and y'. Packing B_{t+1} equals $\max_i B_{t+1} = \|y''\|_1 + |R_{t+1}^0|$. According to the operations an insertion operation yields $\max_i B_{t+1} = \|y'\|_1 + |R_t^0| + 1 = \max_i B'_t + 1$. An insertion operation followed by an union operation yields $\max_i B_{t+1} = \|y'\|_1 + 1 + |R_t^0| + 1 = \max_i B'_t + 2$ and an insertion operation followed by a creation operation yields $\max_i B_{t+1} = \|y'\|_1 + 2 + |R_t^0| - 1 = \max_i B'_t + 1$. Algorithm 2.20 is designed that in the union phase $\max_i B'_t \leq (1+2\Delta)OPT(I_t)+m-2$ since there is an improve call with $\alpha = 2$ and otherwise $\max_i B'_t \leq (1+2\Delta)OPT(I_t)+m-1$ since there is an improve call with $\alpha = 1$. Therefore we have in any case that B_{t+1} uses at most $(1+2\Delta)OPT(I_t)+m \leq (1+2\Delta)OPT(I_{t+1})+m$ bins. The proof that property (2) holds is symmetric since $\|x'\|_1$ increases in the same way as $\|y'\|_1$ and $\|x'\|_1 + |R_t^0| \leq (1+\Delta)OPT(I_t) - \alpha$ for $\alpha = 1$ or $\alpha = 2$. For property (3) note that in the operations a configuration x_i of the fractional solution is increased by 1 if and only if a configuration y_i is increased by 1. Therefore the property that for all configurations $x''_i \leq y''_i$ retains from x' and y'. By Theorem 2.24 the number of non-zero components of x' and y' is bounded by $\Delta OPT(I_t) + m \leq \Delta OPT(I_{t+1}) + m$. By construction of the creation operation and union operation x'' and y'' might have two additional non-zero components. But since these are being reduced by Algorithm 2.13 (note that we increased the number of components being reduced in step 6 by 2), the LP solutions x'' and y'' have at most $\Delta OPT(I_{t+1}) + m$ non-zero components which proves property (4). □

2.4.4 Running Time

Storing items that are in the same rounding group in a heap structure, we can perform each operation (insertion, creation and union) in time $\mathcal{O}(\frac{1}{\epsilon^2}\log(\epsilon^2 t))$. Furthermore Algorithm 2.13 needs to look through all non-zero components. The number of non-zero components is bounded by $\mathcal{O}(\epsilon OPT) = \mathcal{O}(\epsilon t)$. Main part of the complexity lies in finding an approximate LP solution. Let $M(n)$ be the time to solve a system of n linear equations. The running time of max-min resource sharing is then in our case $\mathcal{O}(M(\frac{1}{\epsilon^2})\frac{1}{\epsilon^4} + \frac{1}{\epsilon^7})$ (see [Jan06]).

Therefore the running time of the Algorithm is $\mathcal{O}(M(\frac{1}{\epsilon^2})\frac{1}{\epsilon^4} + \epsilon t + \frac{1}{\epsilon^2}\log(\epsilon^2 t))$.

2.5 Conclusion

Based on approximate solutions, we developed an analog to a theorem of Cook et al. [Coo+86]. Our improvement helps to develop online algorithms with a migration factor that is bounded by a polynomial in $1/\epsilon$, while algorithms based on Cook's theorem usually have exponential migration factors. We therefore applied our techniques to the famous online bin packing problem. This led to the creation of the first fully robust AFPTAS for an NP-hard online optimization problem. The migration factor of our algorithm is of size $\mathcal{O}(\frac{1}{\epsilon^4})$, which is a notable reduction compared to previous robust algorithms. When a new item arrives at time t the algorithm needs running time of $\mathcal{O}(M(\frac{1}{\epsilon^2})\frac{1}{\epsilon^4} + \epsilon t + \frac{1}{\epsilon^2}\log(\epsilon^2 t))$, where $M(n)$ is the time to solve a system of n linear equations. Any improvement to the max-min resource sharing algorithm based on the special structure of bin packing would immediately speed up our online algorithm. We believe that there is room to reduce the running time and the migration factor. Note for example that we give only a very rough bound for the migration factor as the algorithm repacks $\mathcal{O}(\frac{1}{\epsilon^3})$ bins. Repacking these bins in a more carefully way might lead to a smaller migration factor. We mention in closing that the LP/ILP-techniques presented are very general and hence can possibly be used to obtain fully robust algorithms for several other online optimization problems as well.

3 Fully Dynamic Bin Packing Revisited

3.1 Introduction

As in chapter 2 of this thesis we consider the classical bin packing problem where we are given a set I of items with a size function $s\colon I \to (0,1]$ and need to pack them into as few unit sized bins as possible. In practice, the complete instance is often not known in advance, which has lead to the definition of a variety of *online* versions of the bin packing problem. First, in the classical *online bin packing* [Ull71], items arrive over time and have to be packed on arrival. Second, in *dynamic bin packing* [CGJ83], items may also depart over time. This dynamic bin packing model is often used for instance in

- the placement and movement of virtual machines onto different servers for cloud computing [BB10; BKB07; SKZ08; VAN08; Jun+08; Jun+09],
- the development of guaranteed quality of service channels over certain multi-frequency time division multiple access systems [Par+00],
- the placement of processes, which require different resources, onto physical host machines [Sto13; SZ13],
- the resource allocation in a cloud network where the cost depends upon different parameters [DKL14; LTC14].

Third and fourth, as in chapter 2, we may allow already packed items to be slightly rearranged, leading to online bin packing with repacking (known as *relaxed online bin packing*) [GPT00] and dynamic bin packing with repacking (known as *fully dynamic bin packing*) [IL98]. See Figure 3.1 for a short overview on the different models.

Name	Deletion	Repacking
Online Bin Packing	✗	✗
Relaxed Online Bin Packing	✗	✓
Dynamic Bin Packing	✓	✗
Fully Dynamic Bin Packing	✓	✓

Figure 3.1: Overview of online models

The amount of repacking can be measured in different ways. We can either count the total number of moved items at each timestep or the sum of the sizes of the moved items at each timestep. If one wants to count the number of moved items, one typically counts a group of tiny items as a single move. A *shifting move* [GPT00] thus involves either a single large item or a bundle of small items in the same bin of total size s with $1/10 \leq s \leq 1/5$. Such a bundle may consists of up to $\Omega(n)$ (very small) items. If an algorithm measures the repacking by shifting moves, a new tiny item may lead to a large amount of repacking. In order to guarantee that a tiny item i with size $s(i)$ only leads to a small amount of repacking, one may allow to repack items whose size adds up to at most $\beta \cdot s(i)$. The term β is called the *migration factor* [SSS09]. Note that shifting moves and migration factor are incomparable in the sense that a small migration factor does not imply a small number of shifting moves and vice versa.

In order to measure the quality of an online algorithm, we compare the costs incurred by an online algorithm with the costs incurred by an optimal offline algorithm. An *online algorithm* receives as input a *sequence* of items $I = (i_1, i_2, i_3, \ldots)$ and decides at each timestep t, where to place the item i_t without knowing future items $i_{t+1}, i_{t+2}, \ldots$. We denote by $I(t) = (i_1, i_2, \ldots, i_t)$ the instance containing the first t items of the instance I and by $\textsc{opt}(I(t))$ the minimal number of bins needed to pack all items in $I(t)$. Note that the packings corresponding to $\textsc{opt}(I(t))$ and $\textsc{opt}(I(t+1))$ may differ significantly, as those packings do not need to be consistent. For an online algorithm A, we denote by $A(I(t))$ the number of bins generated by the algorithm on the input sequence $I(t)$. Note that A must make its decision online, while $\textsc{opt}(I(t))$ is the optimal value of the offline instance. The quality of an algorithm for the online bin packing problem is typically measured by its *asymptotic competitive ratio*. An online algorithm A is called an *asymptotic α-competitive algorithm*, if there is a function $f \in o(\textsc{opt})$ such that $A(I(t)) \leq \alpha\, \textsc{opt}(I(t)) + f(I(t))$ for all instances I and all $t \leq |I|$. The minimum α such that A is an asymptotic α-competitive algorithm is called the *asymptotic competitive ratio of A*, denoted by $r_\infty^{\text{on}}(A)$, i. e., the ratio is defined as $r_\infty^{\text{on}}(A) = \min\{\alpha \mid A \text{ is an asymptotic } \alpha\text{-competitive algorithm}\}$. The online algorithm A thus has a double disadvantage: It does not know future items and we compare its quality to the optimal offline algorithm which may produce arbitrary different packings at time t and time $t+1$. In order to remedy this situation, one may also compare the solution generated by A to a non-repacking optimal offline algorithm. This non-repacking optimal offline algorithm knows the complete instance, but is not allowed to repack.

In this work, we present new results in fully dynamic bin packing where we measure the quality of an algorithm against a repacking optimal offline algorithm and achieve a asymptotic competitive ratio of $1 + \epsilon$. The amount of repacking is bounded by $\mathcal{O}(1/\epsilon^4 \log(1/\epsilon))$. While we measure the amount of repacking in terms of the migration factor, we also prove that our algorithm uses at most $\mathcal{O}(1/\epsilon^4 \log(1/\epsilon))$ shifting moves. Our algorithm runs in time polynomial in the instance size and in $1/\epsilon$.

3.1.1 Previous Results on Online Variants of Bin Packing

Online Bin Packing

The classical version of online bin packing problem was introduced by Ullman [Ull71]. In this classical model items arrive over time and have to be packed at their arrival, while *one is not allowed to repack already packed items*. Ullman gave the very first online algorithm FirstFit for the problem and proved that it its absolute competitive ratio is at most 2. The next algorithm NextFit was given by Johnson [Joh74], who proved that its absolute competitive is also at most 2. The analysis of the FirstFit algorithm was refined by Johnson, Demers, Ullman, Garey and Graham [Joh+74a], who proved that its asymptotic competitive ratio is at most $17/10$. A revised version of FirstFit, called Revised FirstFit was shown to have asymptotic competitive ratio of at most $5/3$ by Yao [Yao80]. A series of developments of so called *harmonic algorithms* for this problem was started by Lee and Lee [LL85] where algorithms for the online bin packing problem where the competitive ratio was improved step by step. The best known competitive ratio was achieved by Heydrich and van Stee who gave an extension to this the harmonic framework and presented an algorithm with a competitive ratio of 1.5815 for the online bin packing problem. The lower bound on the absolute approximation ratio of $3/2$ also holds for the asymptotic competitive ratio as shown by Yao [Yao80]. This lower bound

was first improved independently by Brown [Bro79] and Liang [Lia80] to 1.53635 and subsequently to 1.54014 by van Vliet [Vli92] and finally to 1.54037 by Balogh, Békési and Galambos [BBG12]. For an extensive survey on online bin packing and various extension of the model we refer to [Cof+13].

Relaxed Online Bin Packing Model

In contrast to the classical online bin packing problem, Gambosi, Postiglione and Talamo [GPT00] considered the online case where one is *allowed to repack items*. They called this model the *relaxed online bin packing model* and proved that the lower bound on the competitive ratio in the classical online bin packing model can be beaten. They presented an algorithm that uses 3 *shifting moves* and has an asymptotic competitive ratio of at most $3/2$, and an algorithm that uses at most 7 shifting moves and has an asymptotic competitive ratio of $4/3$. In another work, Ivković and Lloyd [IL97] gave an algorithm that uses $\mathcal{O}(\log n)$ *amortized* shifting moves and achieves an asymptotic competitive ratio of $1 + \epsilon$. In this amortized setting, shifting moves can be saved up for later use and the algorithm may repack the whole instance sometimes. Epstein and Levin [EL09] used the measure of the migration factor to give an algorithm that has an asymptotic competitive ratio of $1 + \epsilon$ and a migration factor of $2^{\mathcal{O}((1/\epsilon)\log^2(1/\epsilon))}$. We improved upon this result in chapter 2 of this thesis. As shown, our algorithm uses only a polynonomial migration factor of $\mathcal{O}(1/\epsilon^4)$ to achieve an asymptotic competitive ratio of $1 + \epsilon$.

Concerning lower bounds on the migration factor, Epstein and Levin [EL09] showed that no optimal solution can be maintained while having a constant migration factor (independent of $1/\epsilon$). Furthermore, Balogh, Békési, Galambos and Reinelt [Bal+08] proved that a lower bound on the asymptotic competitive ratio of 1.3877 holds, if the amount of repacking is measured by the number of items and one is only allowed to repack a *constant number of items*.

Dynamic Bin Packing

An extension to the classical online bin packing model was given by Coffman, Garey and Johnson [CGJ83], called the *dynamic bin packing* model. In addition to the insertion of items, *items also depart* over time. *No repacking is allowed* in this model. It is easily seen that no algorithm can achieve a constant asymptotic competitive ratio in this setting. In order to measure the performance of an online algorithm A in this case, they compared the *maximum number of bins used by* A with the *maximum number of bins used by an optimal offline algorithm*, i. e., an algorithm A in this dynamic model is called an *asymptotic α-competitive algorithm*, if there is a function $f \in o(\text{max-OPT})$, where $\text{max-OPT}(I) = \max_t \text{OPT}(I(t))$ such that $\max_t A(I(t)) \leq \alpha \cdot \max_t \text{OPT}(I(t)) + f(I)$ for all instances I. The minimum of all such α is called the *asymptotic competitive ratio of* A. Coffman, Garey and Johnson modified the FirstFit algorithm and proved that its asymptotic competitive ratio is at most 2.897. Furthermore, they showed a lower bound of 2.5 on the asymptotic competitive ratio when the performance of the algorithm is compared to a repacking optimal offline algorith, i. e., $\max_t \text{OPT}(I(t))$.

In the case that the performance of the algorithm is compared to an optimal non-repacking offline algorithm, Coffman, Garey and Johnson showed a lower bound of 2.388. This lower bound on the non-repacking optimum was later improved by Chan, Lam and Wong [CLW08] to 2.428 and even further in a later work by Chan, Wong and Yung [CWY09] to 2.5.

Fully Dynamic Bin Packing

We consider the dynamic bin packing when repacking of already packed items is allowed. This model was first investigated by Ivković and Lloyd [IL98] and is called *fully dynamic bin packing*. In this model, items arrive and depart in an online fashion and limited repacking is allowed. The quality of an algorithm is measured by the asymptotic competitive ratio as defined in the classical online model (no maximum is taken as in the dynamic bin packing model). Ivković and Lloyd developed an algorithm that uses amortized $\mathcal{O}(\log n)$ many shifting moves (see definition above) to achieve an asymptotic competitive ratio of $5/4$.

Related Results on the Migration Factor

Since the introduction of the migration factor, several problems were considered in this model and different robust algorithms for these problems have been developed. Following the terminology of Sanders, Sivadasan and Skutella [SSS09] we sometimes use the term *(online) approximation ratio* instead of competitive ratio. Hence, we also use the term asymptotic polynomial time approximation scheme (APTAS) and asymptotic fully polynomial time approximation scheme (AFPTAS) in the context of online algorithms. If the migration factor of an algorithm A only depends upon the approximation ratio ϵ and not on the size of the instance, we say that A *is an robust algorithm*.

In the case of online bin packing, Epstein and Levin [EL09] developed the first robust APTAS for the problem using a migration factor of $2^{\mathcal{O}((1/\epsilon^2)\log(1/\epsilon))}$. They also proved that there is no online algorithm for this problem that has a constant migration factor and that maintains an optimal solution. The APTAS by Epstein and Levin was later improved by by our work that we presented in chapter 2. Recall that we developed a robust AFPTAS for the problem with migration factor $\mathcal{O}(1/\epsilon^4)$. It was shown by Epstein and Levin [EL13] that their APTAS for bin packing can be generalized to packing d-dimensional cubes into unit cubes. Sanders, Sivadasan and Skutella [SSS09] developed a robust polynomial time approximation scheme (PTAS) for the scheduling problem on identical machines with a migration factor of $2^{\mathcal{O}((1/\epsilon)\log^2(1/\epsilon))}$.

Skutella and Verschae [SV16] studied the problem of maximizing the minimum load given n jobs and m identical machines. They also considered a fully dynamic setting, where jobs may depart. They showed that there is no robust PTAS for this machine covering problem with constant migration. The main reason for the nonexistence is due to very small jobs. However, by using an amortized migration factor, they were able to obtain a PTAS for the problem with amortized migration of $2^{\mathcal{O}((1/\epsilon)\log^2(1/\epsilon))}$. Their techniques could also be applied to the scheduling problem considered by Sanders, Sivadasan and Skutella [SSS09].

3.1.2 Our Contributions

Main Result

In this work, we investigate the *fully dynamic bin packing* model. We measure the amount of repacking by the *migration factor*; but our algorithm uses a bounded number of shifting moves as well. Since the work of Ivković and Lloyd from 1998 [IL98], no progress was made on the fully dynamic bin packing problem concerning the asymptotic competitive ratio of $5/4$. It was also unclear whether the number of shifting moves (respectively migration factor) must depend on the number of packed items n. In this chapter we give positive

answers for both of these concerns. We develop an algorithm that provides at each time step t an approximation guarantee of $(1+\epsilon)\,\text{OPT}(I(t)) + \mathcal{O}(1/\epsilon \log(1/\epsilon))$. The algorithm uses a migration factor of $\mathcal{O}(1/\epsilon^4 \cdot \log(1/\epsilon))$ by repacking at most $\mathcal{O}(1/\epsilon^3 \cdot \log(1/\epsilon))$ bins. Hence, the generated solution can be arbitrarily close to the optimum solution, and for every fixed ϵ the provided migration factor is constant (it does not depend on the number of packed items). The running time is polynomial in n and $1/\epsilon$. In case that no deletions are used, the algorithm has a migration factor of $\mathcal{O}(1/\epsilon^3 \cdot \log(1/\epsilon))$, which further improves the migration factor of $\mathcal{O}(1/\epsilon^4)$ that we presented in chapter 2 of this thesis. Since the number of repacked bins is bounded, so is the number of shifting moves as it requires at most $O(1/\epsilon)$ shifting moves to repack a single bin. Furthermore, we prove that there is no asymptotic approximation scheme for the online bin packing problem with a migration factor of $o(1/\epsilon)$ even in the case that no items depart (and even if $\mathcal{P} = \mathcal{NP}$).

Technical Contributions

We use the following techniques to achieve our results:

- In order to obtain a lower bound on the migration factor in Section 3.2, we construct a series of instances that provably need a migration factor of $\Omega(1/\epsilon)$ in order to have an asymptotic approximation ratio of $1+\epsilon$.
- In Section 3.3, we show how to handle large items in a fully dynamic setting. The fully dynamic setting involves more difficulties in the rounding procedure, in contrast to the setting where large items may not depart, treated in chapter 2. A simple adaption of the dynamic techniques developed in chapter 2 does not work (see introduction of Section 3.3). We modify the offline rounding technique by Karmarkar and Karp [KK82] such that a feasible rounding structure can be maintained when items are inserted or removed. This way, we can make use of the LP-techniques developed in chapter 2.
- In Section 3.4, we explain how to deal with small items in a dynamic setting. In contrast to the setting where departure of items is not allowed, the fully dynamic setting provides major challenges in the treatment of small items. An approach is thus developed where small items of similar size are packed near each other. We describe how this structure can be maintained as new items arrive or depart. Note that the algorithm of Ivković and Lloyd [IL98] relies on the ability to manipulate up to $\Omega(n)$ very small items in constant time. See also their updated work for a thorough discussion of this issue [IL09].
- In order to unify the different approaches for small and large items, in Section 3.4.2, we develop an advanced structure for the packing. We give novel techniques and ideas to manage this mixed setting of small and large items. The advanced structure makes use of a potential function, which bounds the number of bins that need to be reserved for incoming items.

3.2 Lower Bound

We start by showing that there is no robust (asymptotic) approximation scheme for bin packing with migration factor of $o(1/\epsilon)$, even if $\mathcal{P} = \mathcal{NP}$. This improves the lower bound given by Epstein and Levin [EL09], which states that no algorithm for bin packing, that maintains an optimal solution can have a constant migration factor. Previously it was

not clear whether there exists a robust approximation algorithm for bin packing with sublinear migration factor or even a constant migration factor.

Theorem 3.1. *For a fixed migration factor $\gamma > 0$, there is no robust approximation algorithm for bin packing with asymptotic approximation ratio better than $1 + \frac{1}{6\lceil\gamma\rceil+5}$.*

Proof. Let $\mathcal{A}$ be an approximation algorithm with migration factor $\gamma > 0$ and $c = \lceil\gamma\rceil$. We will now construct an instance such that the asymptotic approximation ratio of $\mathcal{A}$ with migration factor c is at least $1 + \frac{1}{6c+5}$. The instance contains only two types of items: An A-item has size $a = \frac{3/2}{3c+2}$ and an B-item has size $b = 1/2 - a/3$. For a $M \in \mathbb{N}$, let

$$I_M = [\underbrace{(b, \text{Insert}), (b, \text{Insert}), \ldots, (b, \text{Insert})}_{2M}, \underbrace{(a, \text{Insert}), (a, \text{Insert}), \ldots, (a, \text{Insert})}_{2M(c+1)}]$$

be the instance consisting of $2M$ insertions of B-items, followed by $2M(c+1)$ insertions of A-items. Denote by $r(t)$ the approximation ratio of the algorithm at time $t \in \mathbb{N}$. The approximation ratio of the algorithm is thus $r = \max_t\{r(t)\}$.

The insertion of the B-items produces a packing with β_1 bins containing a single B-item and β_2 bins containing two B-items. These are the only possible packings and hence $\beta_1 + 2\beta_2 = 2M$. The optimal solution is reached if $\beta_1 = 0, \beta_2 = M$. We thus have an approximation ratio of

$$r(2M) =: r_1 = \frac{\beta_1 + \beta_2}{M} = \frac{2M - \beta_2}{M},$$

which is strictly monotonically decreasing in β_2.

The A-items, which are inserted afterwards, may either be put into bins which only contain A-items or into bins which contain only one B-item. The choice of a, b implies $2 \cdot b + a > 1$ which shows that no A-item can be put into a bin containing two B-items. Denote by α the number of bins containing only A-items. The existing B-items may not be moved as the choice of a, b implies $b > c \cdot a > \gamma \cdot a$. At most $\frac{1/2+a/3}{a} = c + 1$ items of type A may be put into the bins containing only one B-item. Note that this also implies that a bin which contains one B-item and $c + 1$ items of type A is filled completely. The optimal packing thus consists of $2M$ of those bins and the approximation ratio of the solution is given by

$$r(2M(c+2)) =: r_2 = \frac{\beta_1 + \beta_2 + \alpha}{2M} = \frac{2M - 2\beta_2 + \beta_2 + \alpha}{2M} = \frac{2M - \beta_2 + \alpha}{2M}.$$

There are at most $\beta_1 \cdot (c+1)$ items of type A which can be put into bins containing only one B-item. The remaining $(2M - \beta_1)(c+1)$ items of type A therefore need to be put into bins containing only A-items. We can thus conclude $\alpha \geq (2M - \beta_1)(c+1)a = (2M - 2M + 2\beta_2)(c+1)a = 2\beta_2(c+1)a$. As noted above, $\frac{1/2+a/3}{a} = c + 1$ and thus $(c+1)a = 1/2 + a/3$. Hence the approximation ratio is at least

$$r_2 = \frac{\beta_1 + \beta_2 + \alpha}{2M} \geq \frac{2M - \beta_2 + 2\beta_2(1/2 + a/3)}{2M} = \frac{2M + \beta_2(-1 + 1 + 2a/3)}{2M} = \frac{2M + \beta_2 \cdot 2a/3}{2M},$$

which is strictly monotonically increasing in β_2.

As $r \geq \max\{r_1, r_2\}$, a lower bound on the approximation ratio is thus given if $r_1 = r_2$ by $\frac{2M-\beta}{M} = \frac{2M+\beta \cdot 2a/3}{2M}$ for a certain β. Solving this equation leads to $\beta = \frac{M}{a/3+1}$. The lower bound is thus given as

$$r \geq \frac{2M-\beta}{M} = 2 - \frac{1}{a/3+1} = 1 + \frac{1}{6c+5}$$

by the choice of a. Note that this lower bound is independent from M. Hence, r is also a lower bound on the asymptotic approximation ratio of any algorithm as the instance size grows with M. □

We obtain the following corollary:

Corollary 3.2. *There is no robust/dynamic (asymptotic) approximation scheme for bin packing with a migration factor* $\gamma \leq 1/6(1/\epsilon - 11) = \Theta(1/\epsilon)$.

3.3 Dynamic Rounding

The goal of this section is to give a robust AFPTAS for the case that only large items arrive and depart. In the first subsection we present a general rounding structure. In the second subsection we give operations on how the rounding can be modified such that the general structure is preserved. We give the final algorithm in Section 3.3.3, which is performed, when large items arrive or depart. Finally, the correctness is proved by using the linear program (LP)/integer linear program (ILP) techniques that we developed in chapter 2.

In chapter 2 of this thesis we developed a dynamic rounding technique based on an offline rounding technique from Fernandez de la Vega and Lueker [FL81]. However, a simple adaption of these techniques does not work in the dynamic case where items may also depart. In the case of the offline rounding by Fernandez de la Vega and Lueker, items are sorted and then collected in groups of the same cardinality. As a new item arrives in an online fashion, this structure can be maintained by inserting the new item to its corresponding group. By shifting the largest item of each group to the left, the cardinality of each group (except for the first one) can be maintained. However, shifting items to the right whenever an item departs leads to difficulties in the LP/ILP techniques. As the rounding for a group may increase, patterns of the existing LP/ILP solution might become infeasible. We overcome these difficulties by developing a new dynamic rounding structure and operations based on the offline rounding technique by Karmarkar and Karp [KK82]. We felt that the dynamic rounding technique based on Karmarkar and Karp is easier to analyze since the structure can essentially be maintained by shifting items.

A bin packing instance consists of a set of *items* $I = \{i_1, i_2, \ldots, i_n\}$ with *size function* $s : I \to [0,1] \cap \mathbb{Q}$. A feasible solution is a partition $B^1, \ldots, B^k$ of I such that $\sum_{i \in B^j} s(i) \leq 1$ for $j = 1, \ldots, k$. We call a partition $B^1, \ldots, B^k$ a *packing* and a single set B^j is called a *bin*. The goal is to find a solution with a minimal number of bins. If the item i is packed into the bin B^j, we write $B(i) = j$. The smallest value of $k \in \mathbb{N}$ such that a packing with k bins exists is denoted by $\textsc{opt}(I, s)$ or if the size function is clear by $\textsc{opt}(I)$. A trivial lower bound is given by the value $\textsc{size}(I, s) = \sum_{i \in I} s(i)$.

3.3.1 Rounding

To obtain an LP formulation of fixed (independent of $|I|$) dimension, we use a rounding technique based on the offline AFPTAS by Karmarkar and Karp [KK82]. In order to use the technique for our dynamic setting, we give a more general rounding. This generalized rounding has a certain structure that is maintained throughout the algorithm and guarantees an approximate solution for the original instance. First, we divide the set of items into *small* ones and *large* ones. An item i is called *small* if $s(i) < \epsilon/14$, otherwise it is called *large*. Instance I is partitioned accordingly into a set of large items I_L and a set of small items I_S. We treat small items and large items differently. Small items can be packed using an algorithm presented in Section 3.4.1 while large items will be assigned using an ILP. In this section we discuss how to handle large items.

First, we characterize the set of large items more precisely by their sizes. We say that two large items i, i' are in the same size category if there is a $\ell \in \mathbb{N}$ such that $s(i) \in (2^{-(\ell+1)}, 2^{-\ell}]$ and $s(i') \in (2^{-(\ell+1)}, 2^{-\ell}]$. Denote the set of all size categories by W. As every large item has size at least $\epsilon/14$, the number of size categories is bounded by $\log(1/\epsilon) + 5$. Next, items of the same size category are characterized by their *block*, which is either A or B and their *position* $r \in \mathbb{N}$ in this block. Therefore, we partition the set of large items into a set of groups $G \subseteq W \times \{A, B\} \times \mathbb{N}$. A group $g \in G$ consists of a triple (ℓ, X, r) with size category $\ell \in W$, block $X \in \{A, B\}$ and position $r \in \mathbb{N}$. The *rounding function* is defined as a function $R : I_L \mapsto G$ that maps each large item $i \in I_L$ to a *group* $g \in G$. By $g[R]$ we denote the set of items being mapped to the group g, i. e., $g[R] = \{i \in I_L \mid R(i) = g\}$.

Let $q(\ell, X)$ be the maximal $r \in \mathbb{N}$ such that $|(\ell, X, r)[R]| > 0$. If (ℓ, X_1, r_1) and (ℓ, X_2, r_2) are two different groups, we say that (ℓ, X_1, r_1) is *left* of (ℓ, X_2, r_1), if $X_1 = A$ and $X_2 = B$ or $X_1 = X_2$ and $r_1 < r_2$. We say that (ℓ, X_1, r_1) is *right* of (ℓ, X_2, r_2) if it is not left of it.

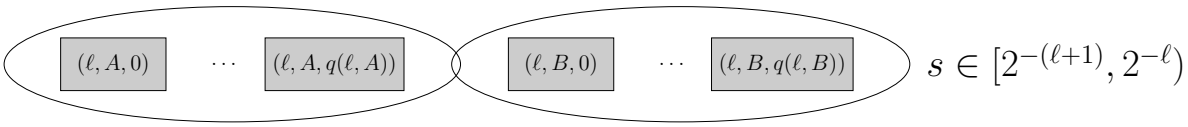


Figure 3.2: Grouping in $(\ell, A, \cdot)$ and $(\ell, B, \cdot)$

Given an instance (I, s) and a rounding function R, we define the rounded size function s^R by rounding the size of every large item $i \in g[R]$ up to the size of the largest item in its group, hence $s^R(i) = \max\{s(i') \mid R(i') = R(i)\}$. We denote by $\text{OPT}(I, s^R)$ the value of an optimal solution of the rounded instance (I, s^R).

Depending on a parameter k, we define the following properties for a rounding function R.

(a) For each $i \in (\ell, X, r)[R]$ we have $2^{-(\ell+1)} < s(i) \leq 2^{-\ell}$.

(b) For each $i \in (\ell, X, r)[R]$ and each $i' \in (\ell, X, r')[R]$ and $r < r'$, we have $s(i) \geq s(i')$.

(c) For each $\ell \in W$ and $1 \leq r \leq q(\ell, A)$ we have $|(\ell, A, r)[R]| = 2^\ell k$ and $|(\ell, A, 0)[R]| \leq 2^\ell k$.

(d) For each $\ell \in W$ and each $0 \leq r \leq q(\ell, B) - 1$ we have $|(\ell, B, r)[R]| = 2^\ell(k-1)$ and furthermore $|(\ell, B, q(\ell, B))[R]| \leq 2^\ell(k-1)$.

Property (a) guarantees that the items are categorized correctly according to their sizes. Property (b) guarantees that items of the same size category are sorted by their size and properties (c) and (d) define the number of items in each group.

Lemma 3.3. *For* $k = \left\lfloor \frac{\text{SIZE}(I_L)\cdot\epsilon}{2(\lfloor\log(1/\epsilon)\rfloor+5)} \right\rfloor$ *the number of non-empty groups in* G *is bounded from above by* $\mathcal{O}(1/\epsilon \log(1/\epsilon))$ *assuming that* $\text{SIZE}(I_L) > 8/\epsilon \cdot (\lceil\log(1/\epsilon)\rceil + 5)$.

Proof. Using the definition of k and the assumption, we show $\frac{2\,\text{SIZE}(I_L)}{k-1} \leq 8/\epsilon(\lfloor\log(1/\epsilon)\rfloor + 5)$. We have

$$\frac{2\,\text{SIZE}(I_L)}{k-1} = \frac{2\,\text{SIZE}(I_L)}{\left\lfloor \frac{\text{SIZE}(I_L)\cdot\epsilon}{2(\lfloor\log(1/\epsilon)\rfloor+5)} \right\rfloor - 1} \leq \frac{2\,\text{SIZE}(I_L)}{\frac{\text{SIZE}(I_L)\cdot\epsilon}{2(\lfloor\log(1/\epsilon)\rfloor+5)} - 2} =$$
$$\frac{2\,\text{SIZE}(I_L)}{\frac{\text{SIZE}(I_L)\cdot\epsilon - 4(\lfloor\log(1/\epsilon)\rfloor+5)}{2(\lfloor\log(1/\epsilon)\rfloor+5)}} = \frac{2\,\text{SIZE}(I_L)\cdot 2(\lfloor\log(1/\epsilon)\rfloor + 5)}{\text{SIZE}(I_L)\cdot\epsilon - 4(\lfloor\log(1/\epsilon)\rfloor + 5)}$$

As $\text{SIZE}(I_L) > 8/\epsilon \cdot (\lceil\log(1/\epsilon)\rceil + 5)$, we have $\epsilon/2\,\text{SIZE}(I_L) > 4(\lfloor\log(1/\epsilon)\rfloor + 5)$. We can thus bound:

$$\frac{2\,\text{SIZE}(I_L)\cdot 2(\lfloor\log(1/\epsilon)\rfloor + 5)}{\text{SIZE}(I_L)\cdot\epsilon - 4(\lfloor\log(1/\epsilon)\rfloor + 5)} \leq \frac{2\,\text{SIZE}(I_L)\cdot 2(\lfloor\log(1/\epsilon)\rfloor + 5)}{\text{SIZE}(I_L)\cdot\epsilon - \epsilon/2\,\text{SIZE}(I_L)} =$$
$$\frac{2\,\text{SIZE}(I_L)\cdot 2(\lfloor\log(1/\epsilon)\rfloor + 5)}{\text{SIZE}(I_L)\cdot\epsilon/2} = \frac{4(\lfloor\log(1/\epsilon)\rfloor + 5)}{\epsilon/2} = \frac{8(\lfloor\log(1/\epsilon)\rfloor + 5)}{\epsilon}$$

Note that property (c) and property (d) imply $|I(\ell)| \geq (q(\ell,A) + q(\ell,B) - 2)2^{\ell}(k-1)$. Hence property (a) implies that $\text{SIZE}(I(\ell), s) \geq |I(\ell)|2^{-(\ell+1)} \geq (q(\ell,A) + q(\ell,B) - 2)(k-1)/2$ and therefore $q(\ell,A) + q(\ell,B) \leq 2\,\text{SIZE}(I(\ell))/(k-1) + 2$. We can now bound the total number of used groups by

$$\sum_{\ell\in W} q(\ell,A) + q(\ell,B) \leq \sum_{\ell\in W}\left(\frac{2\,\text{SIZE}(I(\ell))}{k-1} + 2\right)$$
$$= 2|W| + \frac{2}{k-1}\sum_{\ell\in W}\text{SIZE}(I(\ell)) = 2|W| + \frac{2}{k-1}\,\text{SIZE}(I_L)$$
$$\leq 2|W| + \frac{8}{\epsilon}(\lfloor\log(1/\epsilon)\rfloor + 5) \leq$$
$$2\cdot(\log(1/\epsilon) + 5) + \frac{8}{\epsilon}(\log(1/\epsilon) + 5) =$$
$$(8/\epsilon + 2)(\log(1/\epsilon) + 5) \in \mathcal{O}(1/\epsilon\log(1/\epsilon))$$

The total number of used groups is therefore bounded by $\mathcal{O}(1/\epsilon\log(1/\epsilon))$.

□

The following lemma shows that the rounding function does in fact yield a $(1+\epsilon)$-approximation.

Lemma 3.4. *Given an instance* (I,s) *with items greater than* $\epsilon/14$ *and a rounding function* R *fulfilling properties (a) to (d), then* $\text{OPT}(I, s^R) \leq (1+\epsilon)OPT(I,s)$.

Proof. As (I,s) only contains large items, $I_L = I$. Define for every ℓ the instances $J_\ell = \bigcup_{r=2}^{q(\ell,A)}(\ell,A,r)[R] \cup \bigcup_{r=0}^{q(\ell,B)}(\ell,B,r)[R]$, $J = \bigcup_{\ell\in W} J_\ell$ and $K = \bigcup_{\ell\in W}(\ell,A,0)[R] \cup (\ell,A,1)[R]$. We will now prove, that the error generated by this rounding is bounded by ϵ. As each solution to $J\cup K$ yields a solution to J and a solution to K, we get $\text{OPT}(J\cup K, s^R) \leq \text{OPT}(J, s^R) + \text{OPT}(K, s^R)$. For $i \in (\ell,A,0)[R] \cup (\ell,A,1)[R]$, we have $s(i) \leq \max\{s(i') \mid i' \in (\ell,A,0)[R]\} \leq 2^{-\ell}$ because of property (a). We can therefore pack

at least 2^ℓ items from $(\ell, A, 0)[R] \cup (\ell, A, 1)[R]$ into a single bin. Hence, we get with property (c):

$$\begin{aligned}
&\textsc{opt}((\ell, A, 0)[R] \cup (\ell, A, 1)[R]), s^R) \\
&\leq (|(\ell, A, 0)[R]| + |(\ell, A, 1)[R]|) \cdot 2^{-\ell} \\
&= 2k
\end{aligned}$$

We can therefore bound $\textsc{opt}(K, s^R)$ as follows:

$$\begin{aligned}
\textsc{opt}(K, s^R) &\leq \sum_{\ell \in W} \textsc{opt}((\ell, A, 0)[R] \cup (\ell, A, 1)[R]), s^R) \\
&\leq \sum_{\ell \in W} 2k \\
&\leq 2(\lfloor \log(1/\epsilon) \rfloor + 5)k \\
&= 2\lfloor \frac{\textsc{size}(I)\epsilon}{2(\lfloor \log(1/\epsilon) \rfloor + 5)} \rfloor \cdot (\lfloor \log(1/\epsilon) \rfloor + 5) \\
&\leq 2\frac{\textsc{size}(I)\epsilon}{2(\lfloor \log(1/\epsilon) \rfloor + 5)} \cdot (\lfloor \log(1/\epsilon) \rfloor + 5) \\
&= \epsilon \, \textsc{size}(I) \\
&\leq \epsilon \, \textsc{opt}(I, s)
\end{aligned}$$

Using property (b) for each item in $((\ell, X, r+1)[R]), s^R)$ we find a unique larger item in $(\ell, X, r)[R]$. Therefore we have for every item in the rounded instance (J, s^R) an item with larger size in instance (I, s) and hence

$$\textsc{opt}(J, s^R) \leq \textsc{opt}(I, s).$$

The optimal value of the rounded solution can be bounded by

$$\textsc{opt}(I, s^R) \leq \textsc{opt}(J, s^R) + \textsc{opt}(K, s^R) \leq (1 + \epsilon) \, \textsc{opt}(I, s).$$

□

We therefore have a rounding function, which generates only $\mathcal{O}(1/\epsilon \log(1/\epsilon))$ different item sizes and the generated error is bounded by ϵ.

3.3.2 Rounding Operations

Let us consider the case where large items arrive and depart in an online fashion. Formally this is described by a sequence of pairs $(i_1, A_1), \ldots, (i_n, A_n)$ where $A_i \in \{\text{Insert}, \text{Delete}\}$. At each time $t \in \{1, \ldots, n\}$ we need to pack the item i_t into the corresponding packing of $i_1, \ldots, i_{t-1}$ if $A_i = \text{Insert}$ or remove the item i_t from the corresponding packing of $i_1, \ldots, i_{t-1}$ if $A_i = \text{Delete}$. We will denote the instance $i_1, \ldots, i_t$ at time t by $I(t)$ and the corresponding packing by B_t. We will also round our items and denote the rounding function at time t by R_t. The large items of $I(t)$ are denoted by $I_L(t)$. At time t we are allowed to repack several items with a total size of $\beta \cdot s(i_t)$ but we intend to keep the migration factor β as small as possible. The term $\text{repack}(t) = \sum_{i, B_{t-1}(i) \neq B_t(i)} s(i)$ denotes the sum of the items which are moved at time t, the *migration factor* β of an algorithm is

then defined as $\max_t \{{}^{\text{repack}(t)}/_{s(i_t)}\}$. As the value of SIZE will also change over the time, we define the value $\kappa(t)$ as

$$\kappa(t) = \frac{\text{SIZE}(I_L(t)) \cdot \epsilon}{2(\lfloor \log({}^1/_\epsilon) \rfloor + 5)}.$$

As shown in Lemma 3.3, we will make use of the value $k(t) := \lfloor \kappa(t) \rfloor$.

We present operations that modify the current rounding R_t and packing B_t with its corresponding LP/ILP solutions to give a solution for the new instance $I(t+1)$. At every time t the rounding R_t maintains properties (a) to (d). Therefore the rounding provides an asymptotic approximation ratio of $1+\epsilon$ (Lemma 3.4) while maintaining only $\mathcal{O}({}^1/_\epsilon \log({}^1/_\epsilon))$ many groups (Lemma 3.3). We will now present a way how to adapt this rounding to a dynamic setting, where items arrive or depart online.

Our rounding R_t is manipulated by different *operations*, called the *insert, delete, shiftA* and *shiftB* operation. Some ideas behind the operations are inspired by Epstein and Levin [EL09]. The insert operation is performed whenever a large item arrives and the delete operation is performed whenever a large item departs. The shiftA/shiftB operations are used to modify the number of groups that are contained in the A and B block. As we often need to filter the largest items of a group g belonging to a rounding R, we denote this item by $\lambda(g, R)$.

- shift: A shift operation takes two groups (ℓ, X_1, r_1) and (ℓ, X_2, r_2), where (ℓ, X_1, r_1) is left of (ℓ, X_2, r_2), and a rounding function R and produces a new rounding function R' and packing B' by shifting the largest item from (ℓ, X_2, r_2) to $(\ell, X_2, r_2 - 1)$ and so on until (ℓ, X_1, r_1) is reached.
  - For all groups g left of (ℓ, X_1, r_1) or right of (ℓ, X_2, r_2) set $g[R'] = g[R]$.
  - As we move an item out of (ℓ, X_2, r_2), set

    $$(\ell, X_2, r_2)[R'] = (\ell, X_2, r_2)[R] \setminus \lambda((\ell, X_2, r_2), R).$$

  - As we move an item into (ℓ, X_1, r_1), set

    $$(\ell, X_1, r_1)[R'] = (\ell, X_1, r_1)[R] \cup \lambda(\text{RIGHT}(\ell, X_1, r_1), R).$$

  Whenever a shift-operation on (ℓ, X_1, r_1) and (ℓ, X_2, r_2) is performed, the LP solution x and the corresponding ILP solution y is updated to x' and y'. Let C_i be a configuration containing $\lambda((\ell, X_2, r_2), R)$ with $x_i \geq 1$. Let $C_j = C_i \setminus s(\lambda((\ell, X_2, r_2), R))$ be the configuration without $\lambda((\ell, X_2, r_2), R)$. Set $x'_j = x_j + 1$, $y'_j = y_j + 1$ and $x'_i = x_i - 1$, $y'_i = y_i - 1$. In order to add the new item in (ℓ, X_1, r_1), set $x'_h = x_h + 1$ and $y'_h = y_h + 1$ for the index h with $C_h = \{1 : s(\lambda((\ell, X_1, r_1), R))\}$. The remaining configurations do not change.

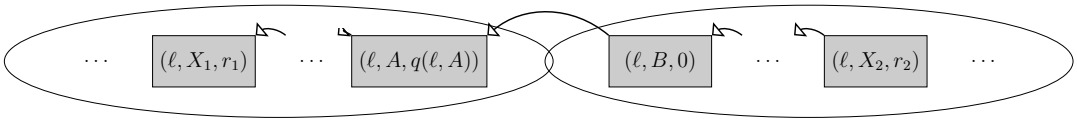


Figure 3.3: shift with parameters (ℓ, X_1, r_1) and (ℓ, X_2, r_2)

- Insert: To insert item i_t, find the corresponding group (ℓ, X, r) with

– $s(i_t) \in (2^{-(\ell+1)}, 2^{-\ell}]$,
– $\min\{s(i) \mid i \in (\ell, X, r-1)\} > s(i_t)$ and
– $s(\lambda((\ell, X, r+1), R)) \leq s(i_t)$.

We will then insert i_t into (ℓ, X, r) and get the rounding R' by shifting the largest element of (ℓ, X, r) to $(\ell, X, r-1)$ and the largest item of $(\ell, X, r-1)$ to $(\ell, X, r-2)$ and so on until $(\ell, A, 0)$ is reached. Formally, set $R^*(i_t) = (\ell, X, r)$ and $R^*(i_j) = R(i_j)$ for $j \neq t$. The rounding function R' is then obtained by applying the shift operation on R^* i.e. the new rounding is $R' = \text{shift}((\ell, A, 0), (\ell, X, r), R^*)$.

In order to pack the new item, let i be the index with $C_i = \{1 : s(\lambda((\ell, X, r), R'))\}$, as i_t is rounded to the largest size in $(\ell, X, r)[R]$ after the shift. Place item i_t into a new bin by setting $B'(i_t) = \max_j B(i_j) + 1$ and $x'_i = x_i + 1$ and $y'_i = y_i + 1$.

If $|(\ell, A, 0)[R']| = 2^\ell \cdot k + 1$, we have to create a new rounding group $(\ell, A, -1)$. Additionally we shift the largest item in $(\ell, A, 0)[R']$ to the new group $(\ell, A, -1)[R']$. The final rounding R'' is then obtained by setting $(\ell, A, r)[R''] = (\ell, A, r-1)[R']$ i.e. incrementing the number of each rounding group by 1. Note that the largest item in $(\ell, A, 0)[R']$ is already packed into a bin of its own due to the shift operation. Hence, no change in the packing or the LP/ILP solution is needed. The insert operation thus yields a new packing B' (or B'') which uses two more bins than the packing B.

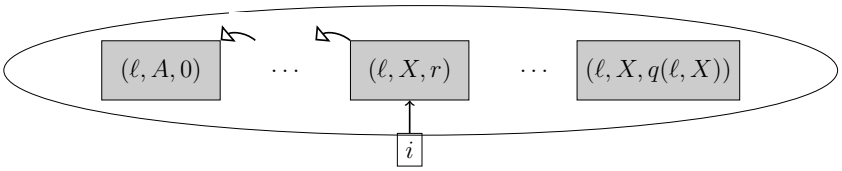


Figure 3.4: Insert i into $(\ell, X, \cdot)$

- Delete: To delete item i_t from the group (ℓ, X, r) with $R(i_t) = (\ell, X, r)$, we remove i_t from this group and move the largest item from $(\ell, X, r+1)$ into (ℓ, X, r) and the largest item from $(\ell, X, r+2)$ into $(\ell, X, r+1)$ and so on until $(\ell, B, q(\ell, B))$. Formally the rounding R' is described by the expression $\text{shift}((\ell, X, r), (\ell, B, q(\ell, B)), R^*)$ where

$$g[R^*] = \begin{cases} (\ell, X, r)[R] \setminus \{i_t\} & g = (\ell, X, r) \\ g[R] & \text{else} \end{cases}.$$

  As a single shift operation is used, the delete operation yields a new packing B' which uses one more bin than the packing B.

  For the LP/ ILP solution let C_i be a configuration containing $\lambda((\ell, B, q(\ell, B)), R)$ with $x_i \geq 1$. Let $C_j = C_i \; s(\lambda((\ell, B, q(\ell, B)), R))$ be the configuration without the item $\lambda((\ell, B, q(\ell, B)), R)$. Set $x'_j = x_j + 1$, $y'_j = y_j + 1$ and $x'_i = x_i - 1$, $y'_i = y_i - 1$. Set $B'(i_j) = B(i_j)$ for all $j \neq t$ in order to remove the item i_t from the packing.

To control the number of groups in A and B we introduce operations shiftA and shiftB that increase or decrease the number of groups in A respectively B. An operation shiftA increases the number of groups in A by 1 and decreases the number of groups in B by 1. Operation shiftB is does the inverse of shiftA.

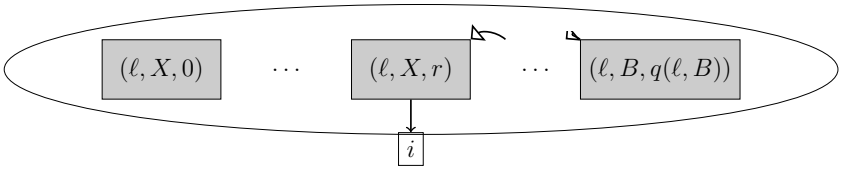


Figure 3.5: Delete i from $(\ell, X, \cdot)$

- shiftA: In order to move a group from B to A we will perform exactly 2^ℓ times the operation $\text{shift}((\ell, B, 0), (\ell, B, q(\ell, B)), R)$ to receive the rounding R^*. Instead of opening a new bin for each of those 2^ℓ items in every shift operation, we rather open one bin containing all items. Since every item in the corresponding size category has size $\leq 2^{-\ell}$, the items fit into a single bin. The group $(\ell, B, 0)$ has now the same size as the groups in $(\ell, A, \cdot)$. We transfer $(\ell, B, 0)$ to block A. Hence we define for the final rounding R' that $(\ell, A, r)[R'] = (\ell, A, r)[R^*]$ for $r = 0, \ldots, q(\ell, A)$ and $(\ell, A, q(\ell, A) + 1)[R'] = (\ell, B, 0)[R^*]$ as well as $(\ell, B, r)[R'] = (\ell, B, r + 1)[R^*]$ for $r = 0, \ldots, q(\ell, B) - 1$. The resulting packing B' hence uses one more bin than the packing B.

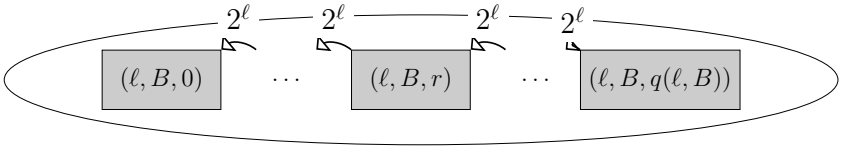


Figure 3.6: shiftA

- shiftB: In order to move a group from A to B we will perform exactly 2^ℓ times the operation $\text{shift}((\ell, A, 0), (\ell, A, q(\ell, A)), R)$ to receive the rounding R^*. As before in shiftA, we open a single bin containing all of the 2^ℓ items. The group $(\ell, A, q(\ell, A))$ has now the same size as the groups in $(\ell, B, \cdot)$. We transfer $(\ell, A, q(\ell, A))$ to block B. Similar to shiftA we define for the final rounding R' that $(\ell, A, r)[R'] = (\ell, A, r)[R^*]$ for $r = 0, \ldots, q(\ell, A) - 1$ and $(\ell, B, 0)[R'] = (\ell, A, q(\ell, A))[R^*]$ as well as $(\ell, B, r + 1)[R'] = (\ell, B, r)[R^*]$. The resulting packing B' hence uses one more bin than the packing B.

Lemma 3.5. *Let R be a rounding function fulfilling properties (a) to (d). Applying one of the operations insert, delete, shiftA or shiftB on R results in a rounding function R' fulfilling properties (a) to (d).*

Proof. Property (a) is always fulfilled as no item is moved between different size categories and the insert operation inserts an item into its appropriate size category.

As the order of items never changes and the insert operation inserts an item into the appropriate place, property (b) also holds.

For properties (c) and (d) we first note that the operation $\text{shift}(g, g', R)$ increases the number of items in g by 1 and decreases the number of items in g' by 1. The insert operation consists of adding a new item to a group g followed by a $\text{shift}((\ell, A, 0), g, R)$ operation. Hence the number of items in every group except for $(\ell, A, 0)$ (which is increased by 1) remains the same. The delete operation consists of removing an item from a group g followed by a $\text{shift}(g, (\ell, B, q(\ell, B)), R)$ operation. Therefore the number of items in all groups except for $(\ell, B, q(\ell, B))$ (which is decreased by 1) remains the same. As the

number of items in $(\ell, A, 0)$ and $(\ell, B, q(\ell, B))$ are treated seperately and may be smaller than $2^\ell \cdot k$ respectively $2^\ell \cdot (k-1)$, the properties (c) and (d) are always fulfilled for the insert and the delete operation. Concerning the shiftA operation we increase the number of items in a group $(\ell, B, 0)$ by 2^ℓ. Therefore it now contains $2^\ell(k-1) + 2^\ell = 2^\ell \cdot k$ items, which equals the number of items in groups of block A. As this group is now moved to block A, the properties (c) and (d) are fulfilled. Symmetrically the shiftB operation decreases the number of items in a group $(\ell, A, q(\ell, A))$ by 2^ℓ. Therefore the number of items in the group is now $2^\ell \cdot k - 2^\ell = 2^\ell \cdot (k-1)$, which equals the number of items in the groups of block B. As this group is now moved to block B, the properties (c) and (d) are fulfilled. □

According to Lemma 3.3 the rounded instance (I, s^R) has $\mathcal{O}(1/\epsilon \log(1/\epsilon))$ different item sizes (given a suitable k). Using the LP formulation of Eisemann [Eis57], the resulting LP called $LP(I, s^R)$ has $m = \mathcal{O}(1/\epsilon \log(1/\epsilon))$ constraints. We say a packing B *corresponds* to a rounding R and an integral solution y of the ILP if all items in (I, s^R) are packed by B according to y.

Lemma 3.6. *Applying any of the operations insert, delete, shiftA or shiftB on a rounding function R and ILP solution y with corresponding packing B defines a new rounding function R' and a new integral solution y'. Solution y' is a feasible solution of $LP(I, s^{R'})$.*

Proof. We have to analyze how the LP for instance $(I, s^{R'})$ changes in comparison to the LP for instance (I, s^R).
Shift Operation: A single shift(g_1, g_2, R) operation moves one item from each group g between g_1 and g_2 into g and one item out of g. As no item is moved out of g_1 and no item is moved into g_2, the number of items in g_1 is increased by 1 and the number of items in g_2 is decreased by 1. The right hand side of the $LP(I, s^R)$ is defined by the cardinalities $|g[R]|$ of the rounding groups g in R. As only the cardinalities of g_1 and g_2 change by ± 1 the right hand side changes accordingly to ± 1 in the corresponding components of y. The moved item from g_2 is removed from the configuration and a new configuration containing the new item of g_1 is added. The LP and ILP solutions x and y are being modified such that $\lambda(g_2, R)$ is removed from its configuration and a new configuration is added such that the enhanced right hand side of g_1 is covered. Since the largest item $\lambda(g, R)$ of every group g between g_1 and g_2 is shifted to its left group, the size $s^{R'}(i)$ of item $i \in g[R]$ is defined by $s^{R'}(i) = s(\iota(g, R))$, where $\iota(g, R)$ is the second largest item of $g[R]$. Therefore each item in $(I, s^{R'})$ is rounded to a smaller or equal value as $s(\iota(g, R)) \leq s(\lambda(g, R))$. All configurations of (I, s^R) can thus be transformed into feasible configurations of $(I, s^{R'})$.
Insert Operation: The insert operation consists of inserting the new item into its corresponding group g followed by a shift operation. Inserting the new item into g increases the right hand side of the LP by 1. To cover the increased right hand side, we add a new configuration $\{1 : s^{R'}(i)\}$ containing only the new item. In order to reflect the change in the LP solution, the new item is added into an additional bin. The remaining changes are due to the shift operation already treated above.
Delete Operation: The delete operation consists of removing an item i from its corresponding group g followed by a shift operation. Removing the new item from g decreases the right hand side of the LP by 1. The current LP and ILP solutions x and y do not need to be changed to cover the new right hand side. The remaining changes are due to the shift operation already treated above.
shiftA/shiftB Operation: As the shiftA and shiftB operations consist only of repeated

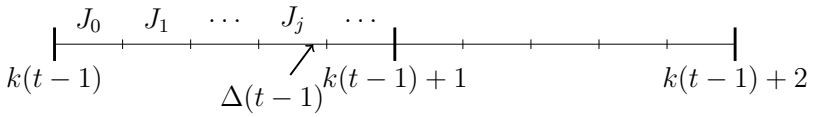
use of the shift operation, the correspondence between the packing and the LP/ILP solution follow simply by induction. □

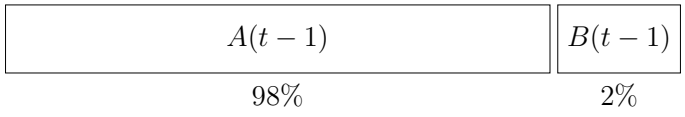
3.3.3 Algorithm for Dynamic Bin Packing

We will use the operations from the previous section to obtain a dynamic algorithm for bin packing with respect to large items. The operations insert and delete are designed to process the input depending of whether an item is to be inserted or removed. Keep in mind that the parameter $k = \lfloor \kappa \rfloor = \left\lfloor \frac{\text{SIZE}(I_L) \cdot \epsilon}{2(\lfloor \log(1/\epsilon) \rfloor + 5)} \right\rfloor$ changes over time as $\text{SIZE}(I_L)$ may increase or decrease. In order to fulfill the properties (c) and (d), we need to adapt the number of items per group whenever k changes. The shiftA and shiftB operations are thus designed to manage the dynamic number of items in the groups as k changes. Note that a group in the A-block with parameter k has by definition the same number of items as a group in the B-block with parameter $k-1$ assuming they are in the same size category. If k increases, the former A block is treated as the new B block in order to fulfill the properties (c) and (d) while a new empty A block is introduced. To be able to rename the blocks, the B block needs to be empty. Accordingly the A block needs to be empty if k decreases in order to treat the old B block as new A block. Hence we need to make sure that there are no groups in the B-block if k increases and vice versa, that there are no groups in the A-block if k decreases.

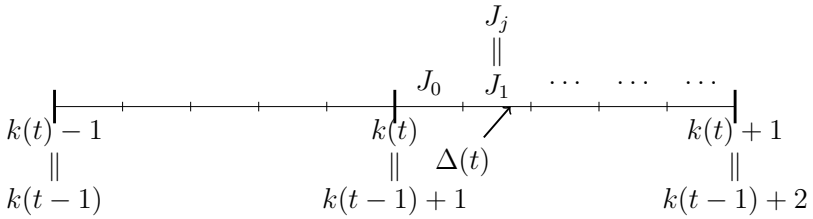
We denote the number of all groups in the A-blocks at time t by $A(t)$ and the number of groups in B-blocks at time t by $B(t)$. To make sure that the B-block (respectively the A-block) is empty when k increases (decreases) the ratio $\frac{A(t)}{A(t)+B(t)}$ needs to correlate to the fractional digits of $\kappa(t)$ at time t denoted by $\Delta(t)$. Hence we partition the interval $[0,1)$ into exactly $A(t)+B(t)$ smaller intervals $J_i = \left[\frac{i}{A(t)+B(t)}, \frac{i+1}{A(t)+B(t)}\right)$. We will make sure that $\Delta(t) \in J_i$ iff $\frac{A(t)}{A(t)+B(t)} \in J_i$. Note that the term $\frac{A(t)}{A(t)+B(t)}$ is 0 if the A-block is empty and the term is 1 if the B-block is empty. This way, we can make sure that as soon as $k(t)$ increases, the number of B-blocks is close to 0 and as soon as $k(t)$ decreases, the number of A-blocks is close to 0. Therefore, the A, B-block can be renamed whenever $k(t)$ changes. The algorithm uses shiftA and shiftB operations to adjust the number of groups in the A- and B-block. Recall that a shiftA operation reduces the number of groups in the B-block by 1 and increases the number of groups in the A-block by 1 (shiftB works vice versa). Let d be the number of shiftA/shiftB operations that need to be performed to adjust $\frac{A(t)}{A(t)+B(t)}$.

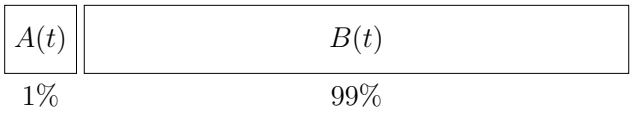
In the following algorithm we make use of an algorithm 2.13 called IMPROVE, which we developed in chapter 2 to reduce the number of used bins. Using IMPROVE(x) on a packing B with approximation guarantee $\max_i B(i) \leq (1+\bar{\epsilon})\,\text{OPT} + C$ for some $\bar{\epsilon} = \mathcal{O}(\epsilon)$ and some additive term C yields a new packing B' with approximation guarantee $\max_i B(i) \leq (1+\bar{\epsilon})\,\text{OPT} + C - x$. We use the operations in combination with the improve algorithm to obtain a fixed approximation guarantee.





(a) Before Insert





(b) After Insert

Figure 3.7: Comparison of the situation before and after an Insert Operation

Algorithm 3.7 (AFPTAS for large items)**.**

Algorithm: *Insertion*

if *SIZE*$(I(t)) < (m+2)(1/\delta+2)$ *or SIZE*$(I(t)) < 8(1/\delta+1)$ **then**
  use offline Bin Packing
else
  IMPROVE*(2); insert(i);*
  `// Shifting to the correct interval`
  Let J_i *be the interval containing* $\Delta(t)$*;*
  Let J_j *be the interval containing* $\frac{A(t)}{A(t)+B(t)}$*;*
  Set $d = i - j$*;*
  if $k(t) > k(t-1)$ **then** `// Modulo` $A(t)+B(t)$ `when` k `increases`
    $d = d + (A(t)+B(t))$*;*
  `// Shifting` d `groups from` B `to` A
  for $p := 0$ *to* $|d|-1$ **do**
    if *i+p = A(t) + B(t)* **then**
      Rename(A, B)*;*
    IMPROVE*(1); shiftA;*

Algorithm: *Deletion*

if *SIZE(I(t))* $< (m+2)(1/\delta+2)$ *or SIZE(I(t))* $< 8(1/\delta+1)$ **then**
  use offline Bin Packing
else
  `// Departing item` i
  IMPROVE*(4); delete(i);*
  REDUCECOMPONENTS*;*
  `//`
  `// Shifting to the correct interval`
  Let J_i *be the interval containing* $\Delta(t)$*;*
  Let J_j *be the interval containing* $\frac{A(t)}{A(t)+B(t)}$*;*
  Set $d = i - j$*;*
  if $k(t) < k(t-1)$ **then** `// Modulo` $A(t)+B(t)$ `when` k `decreases`
    d = d - *(A(t)+B(t));*
  `// Shifting` d `groups from A to B`
  for $p := 0$ *to* $|d|-1$ **do**
    if *i-p = 0* **then**
      Rename(A,B);
    IMPROVE*(3); shiftB;*

Note that as exactly d groups are shifted from A to B (or B to A) we have by definition that $\Delta(t) \in \left[\frac{A(t)}{A(t)+B(t)}, \frac{A(t)+1}{A(t)+B(t)}\right)$ at the end of the algorithm. Note that d can be bounded by 11.

Lemma 3.8. *At most* 11 *groups are shifted from A to B (or B to A) in Algorithm 3.7.*

Proof. Since the value $|\,\text{SIZE}(I(t-1)) - \text{SIZE}(I(t))|$ changes at most by 1 we can bound $D = |\kappa(t-1) - \kappa(t)|$ by $\frac{\epsilon}{2(\lfloor\log(1/\epsilon)\rfloor+5)} \leq \frac{\epsilon}{\log(1/\epsilon)+5}$ to obtain the change in the fractional part. By Lemma 3.3 the number of intervals (=the number of groups) is bounded by $(\frac{8}{\epsilon}+2)(\log(1/\epsilon)+5)$. Using $\Delta(t-1) \in [\frac{A(t-1)}{A(t-1)+B(t-1)}, \frac{A(t-1)+1}{A(t-1)+B(t-1)})$ and the fact that the number of groups $A(t-1)+B(t-1)$ increases or decreases at most by 1, we can give a bound for the parameter d in both cases by

$$d \leq \frac{D}{\text{interval length}} + 1 = D \cdot \#intervals + 1 \leq$$
$$\left((\frac{\epsilon}{\log(1/\epsilon)+5}) \cdot (\frac{8}{\epsilon}+2) \cdot (\log(1/\epsilon)+5)\right) + 1 =$$
$$8 + 2\epsilon + 1 < 11$$

Hence, the number of shiftA/shiftB operations is bounded by 11. □

Lemma 3.9. *Every rounding function* R_t *produced by Algorithm 3.7 fulfills properties* (a) *to* (d) *with parameter* $k(t) = \left\lfloor \frac{\text{SIZE}(I_L)\cdot\epsilon}{2(\lfloor\log(1/\epsilon)\rfloor+5)} \right\rfloor$.

Proof. Since Algorithm 3.7 uses only the operations insert, delete, shiftA and shiftB, the properties (a) to (d) are always fulfilled by Lemma 3.5 and the LP/ILP solutions x, y correspond to the rounding function by Lemma 3.6.

Furthermore, the algorithm is designed such that whenever k increases the B-block is empty and the A-block is renamed to be the new B-block. Whenever k decreases the A-block is empty and the B-block is renamed to be the new A-block. Therefore the number of items in the groups is dynamically adapted to match with the parameter k. □

3.3.4 Large items

In this section we prove that Algorithm 3.7 is a dynamic robust AFPTAS for the bin packing problem if all items have size at least $\epsilon/14$. The treatment of small items is described in Section 3.4 and the general case is described in Section 3.4.2.

We will prove that the migration between packings B_t and B_{t+1} is bounded by $\mathcal{O}(1/\epsilon^3 \log(1/\epsilon))$ and that we can guarantee an asymptotic approximation ratio such that $\max B_t(i) \leq (1+2\Delta)\,\text{OPT}(I(t),s) + \text{poly}(1/\Delta)$ for a parameter $\Delta = \mathcal{O}(\epsilon)$ and for every $t \in \mathbb{N}$. Recall that the Algorithm IMPROVE was developed in chapter 2 of this thesis to improve the objective value of an LP with integral solution y and corresponding fractional solution x. In the following we restate the algorithm IMPROVE 2.13 and the respective theorem 2.14 and corollary 2.15 of chapter 2 with slightly changed notation.

For a vector $z \in \mathbb{R}^n$ let $V(z)$ be the set of all integral vectors $v = (v_1, \dots v_n)^T$ such that $0 \leq v_i \leq z_i$. Let x be an approximate solution of the LP $\min\{\|x\|_1 \mid Ax \geq b, x \geq 0\}$ with m inequalities and let $\|x\|_1 \leq (1+\delta)\,\text{LIN}$ and $\|x\|_1 \geq 2\alpha(1/\delta+1)$, where LIN denotes the fractional optimum of the LP and $\alpha \in \mathbb{N}$ is part of the input of the algorithm (see Chapter 2). Let y be an approximate integer solution of the LP with $\|y\|_1 \leq \text{LIN} + 2C$ for some value $C \geq \delta\,\text{LIN}$ and with $\|y\|_1 \geq (m+2)(1/\delta+2)$. Suppose that both x and y have only $\leq C$ non-zero components. For every component i we suppose that $y_i \geq x_i$. Furthermore we are given indices $a_1, \dots, a_K$, such that the non-zero components y_{a_j} are sorted in non-decreasing order, i. e., $y_{a_1} \leq \dots \leq y_{a_K}$.

Algorithm 3.10 (IMPROVE)**.**

1. *Set* $x^{var} := 2\frac{\alpha(1/\delta+1)}{\|x\|}x$, $x^{fix} := x - x^{var}$ *and* $b^{var} = b - A(x^{fix})$
2. *Compute an approximate solution* $\hat{x}$ *of the LP* $\min\{\|x\|_1 \mid Ax \geq b^{var}, x \geq 0\}$ *with ratio* $(1+\delta/2)$
3. *If* $\left\|x^{fix} + \hat{x}\right\|_1 \geq \|x\|_1$ *then set* $x' = x$, $\hat{y} = y$ *and goto step 9*
4. *Choose the largest* ℓ *such that the sum of the smallest components* $y_1, \dots, y_\ell$ *is bounded by* $\sum_{1 \leq i \leq \ell} y_{a_i} \leq (m+2)(1/\delta+2)$
5. *For all* i *set* $\bar{x}_i^{fix} = \begin{cases} 0 & \text{if } i = a_j, j \leq \ell \\ x_i^{fix} & \text{else} \end{cases}$ *and* $\bar{y}_i = \begin{cases} 0 & \text{if } i = a_j, j \leq \ell \\ y_i & \text{else} \end{cases}$
6. *Set* $\bar{x} = \hat{x} + x_\ell$ *where* x_ℓ *is a vector consisting of the components* $x_{a_1}, \dots, x_{a_\ell}$. *Reduce the number of non-zero components to at most* $m+1$.
7. $x' = \bar{x}^{fix} + \bar{x}$
8. *For all non-zero components* i *set* $\hat{y}_i = \max\{\lceil x'_i \rceil, \bar{y}_i\}$
9. *If possible choose* $d \in V(\hat{y} - x')$ *such that* $\|d\|_1 = \alpha(1/\delta+1)$ *otherwise choose* $d \in V(\hat{y} - x')$ *such that* $\|d\|_1 < \alpha(1/\delta+1)$ *is maximal.*
10. *Return* $y' = \hat{y} - d$

In the following we prove that the algorithm IMPROVE applied to the bin packing ILP actually generates a new improved packing B' from the packing B with corresponding LP and ILP solutions x' and y'. We therefore use Theorem 3.11 and Corollary 3.12 that were proven in chapter 2.

Theorem 3.11. *Let x be a solution of the LP with $\|x\|_1 \leq (1+\delta)\,\text{LIN}$ and furthermore $\|x\|_1 \geq 2\alpha(1/\delta+1)$. Let y be an integral solution of the LP with $\|y\|_1 \leq \text{LIN}+2C$ for some value $C \geq \delta\,\text{LIN}$ and with $\|y\|_1 \geq (m+2)(1/\delta+2)$. Solutions x and y have the same number of non-zero components and for each component we have $x_i \leq y_i$. The Algorithm* IMPROVE(α) *then returns a fractional solution x' with $\|x'\|_1 \leq (1+\delta)\,\text{LIN}-\alpha$ and an integral solution y' where one of the two properties hold: $\|y'\|_1 = \|y\|_1 - \alpha$ or $\|y'\|_1 = \|x'\|_1 + C$. Both, x' and y' have at most C non-zero components and the distance between y' and y is bounded by $\|y'-y\|_1 = \mathcal{O}(\frac{m+\alpha}{\delta})$.*

Corollary 3.12. *Let $\|x\|_1 = (1+\delta')\,\text{LIN}$ for some $\delta' \geq \delta$ and $\|x\|_1 \geq 2\alpha(1/\delta+1)$ and let $\|y\|_1 \leq \text{LIN}+2C$ for some $C \geq \delta'\,\text{LIN}$ and $\|y\|_1 \geq (m+2)(1/\delta+2)$. Solutions x and y have the same number of non-zero components and for each component we have $x_i \leq y_i$. Then Algorithm* IMPROVE(α) *returns a fractional solution x' with $\|x'\|_1 \leq \|x\|_1 - \alpha = (1+\delta')\,\text{LIN}-\alpha$ and integral solution y' where one of the two properties hold: $\|y'\|_1 = \|y\|_1 - \alpha$ or $\|y'\|_1 = \|x\|_1 - \alpha + C$. Both, x' and y' have at most C non-zero components and the distance between y' and y is bounded by $\|y'-y\|_1 = \mathcal{O}(\frac{m+\alpha}{\delta})$.*

Let $\Delta = \epsilon + \delta + \epsilon\delta$ and $C = \Delta\,\text{OPT}(I,s) + m$.

Theorem 3.13. *Given a rounding function R and an LP defined for (I, s^R), let x be a fractional solution of the LP with $\|x\|_1 \leq (1+\Delta)\,\text{OPT}(I,s)$, $\|x\|_1 \geq 2\alpha(1/\delta+1)$ and $\|x\|_1 = (1+\delta')\,\text{LIN}(I,s^R)$ for some $\delta' > 0$. Let y be an integral solution of the LP with $\|y\|_1 \geq (m+2)(1/\delta+2)$ and corresponding packing B such that $\max_i B(i) = \|y\|_1 \leq (1+2\Delta)\,\text{OPT}(I,s)+m$. Suppose x and y have the same number $\leq C$ of non-zero components and for all components i we have $y_i \geq x_i$. Then Algorithm* IMPROVE(α) *on x and y returns a new fractional solution x' with $\|x'\|_1 \leq (1+\Delta)\,\text{OPT}(I,s) - \alpha$ and also a new integral solution y' with corresponding packing B' such that*

$$\max_i B'(i) = \|y'\|_1 \leq (1+2\Delta)\,\text{OPT}(I,s) + m - \alpha.$$

Further, both solutions x' and y' have the same number $\leq C$ of non-zero components and for each component we have $x'_i \leq y'_i$. The number of changed bins from the packing B to the packing B' is bounded by $\mathcal{O}(\frac{m}{\delta})$.

Proof. To use Theorem 3.11 and Corollary 3.12 we have to prove that certain conditions follow from the requisites of Theorem 3.13. We have $\max_i B(i) = \|y\|_1 \leq (1+2\Delta)\,\text{OPT}(I,s)+m$ by condition. Since $\text{OPT}(I,s) \leq \text{OPT}(I,s^R)$ we obtain for the integral solution y that $\|y\|_1 \leq 2\Delta\,\text{OPT}(I,s)+m+\text{OPT}(I,s^R) \leq 2\Delta\,\text{OPT}(I,s)+m+\text{LIN}(I,s^R)+m$. Hence by definition of C we get $\|y\|_1 \leq \text{LIN}(I,s^R)+2C$. This is one requirement to use Theorem 3.11 or Corollary 3.12. We distinguish the cases where $\delta' \leq \delta$ and $\delta' > \delta$ and look at them separately.

Case 1: $\delta' \leq \delta$. For the parameter C we give a lower bound by the inequality $C > \Delta\,\text{OPT}(I,s) = (\delta+\epsilon+\delta\epsilon)\,\text{OPT}(I,s)$. Lemma 3.4 shows that $\text{OPT}(I,s^R) \leq (1+\epsilon)\,\text{OPT}(I,s)$ and therefore yields

$$\begin{aligned}
&\frac{\delta+\epsilon+\delta\epsilon}{1+\epsilon}\,\text{OPT}(I,s^R) = \frac{(1+\delta)(1+\epsilon)-1}{1+\epsilon}\,\text{OPT}(I,s^R)\\
&= (1+\delta)\,\text{OPT}(I,s^R) - \frac{1}{1+\epsilon}\,\text{OPT}(I,s^R)\\
&\geq \delta\,\text{OPT}(I,s^R) \geq \delta LIN(I,s^R)
\end{aligned}$$

and hence $C > \delta \operatorname{LIN}(I, s^R)$. We can therefore use Theorem 3.11.

Algorithm IMPROVE returns by Theorem 3.11 a x' with $\|x'\|_1 \leq (1+\delta)\operatorname{LIN}(I, s^R) - \alpha \leq (1+\delta)\operatorname{OPT}(I, s^R) - \alpha$ and an integral solution y' with $\|y'\|_1 \leq \|x'\|_1 + C$ or $\|y'\|_1 \leq \|y\|_1 - \alpha$. Using that $\operatorname{OPT}(I, s^R) \leq (1+\epsilon)\operatorname{OPT}(I, s)$ we can conclude $\|x'\|_1 \leq (1+\delta)(1+\epsilon)\operatorname{OPT}(I, s) - \alpha = (1+\Delta)\operatorname{OPT}(I, s) - \alpha$. In the case where $\|y'\|_1 \leq \|x'\|_1 + C$ we can bound the number of bins of the new packing B' by $\max_i B'(i) = \|y'\|_1 \leq \|x'\|_1 + C \leq (1+2\Delta)\operatorname{OPT}(I, s) + m - \alpha$. In the case that $\|y'\|_1 \leq \|y\|_1 - \alpha$ we obtain $\max_i B'(i) = \|y'\|_1 \leq \|y\|_1 - \alpha \leq (1+2\Delta)\operatorname{OPT}(I, s) + m - \alpha$. Furthermore we know by Theorem 3.11 that x' and y' have at most C non-zero components.

Case 2: $\delta' > \delta$. First we prove that C is bounded from below. Since $\|x\|_1 = (1+\delta')\operatorname{LIN}(I, s^R) \leq (1+\Delta)\operatorname{OPT}(I, s) = \operatorname{OPT}(I, s) + \Delta\operatorname{OPT}(I, s) \leq \operatorname{OPT}(I, s^R) + \Delta\operatorname{OPT}(I, s) \leq \operatorname{LIN}(I, s^R) + m + \Delta\operatorname{OPT}(I, s) = \operatorname{LIN}(I, s^R) + C$ we obtain that $C \geq \delta'\operatorname{LIN}(I, s^R)$, which is a requirement to use Corollary 3.12. By using Algorithm IMPROVE on solutions x with $\|x\|_1 = (1+\delta')\operatorname{LIN}(I, s^R)$ and y with $\|y\|_1 \leq \operatorname{LIN}(I, s^R) + 2C$ we obtain by Corollary 3.12 a fractional solution x' with $\|x'\|_1 \leq \|x\|_1 - \alpha \leq (1+\Delta)\operatorname{OPT}(I, s) - \alpha$ and an integral solution y' with either $\|y'\|_1 \leq \|y\|_1 - \alpha$ or $\|y'\|_1 \leq \|x\|_1 + C - \alpha$. So for the new packing B' we can guarantee that $\max_i B'(i) = \|y'\|_1 \leq \|y\|_1 - \alpha = \max_i B(i) - \alpha \leq (1+2\Delta)\operatorname{OPT}(I, s) + m - \alpha$ if $\|y'\|_1 \leq \|y\|_1 - \alpha$. In the case that $\|y'\|_1 \leq \|x\|_1 + C - \alpha$, we can guarantee that $\max_i B'(i) = \|y'\|_1 \leq \|x\|_1 + C - \alpha \leq (1+\Delta)\operatorname{OPT}(I, s) + C - \alpha \leq (1+2\Delta)\operatorname{OPT}(I, s) + m - \alpha$. Furthermore we know by Corollary 3.11 that x' and y' have at most C non-zero components.

Theorem 3.11 as well as Corollary 3.12 state that the distance $\|y' - y\|_1$ is bounded by $\mathcal{O}(m/\delta)$. Since y corresponds directly to the packing B and the new integral solution y' corresponds to the new packing B', we know that only $\mathcal{O}(m/\delta)$ bins of B need to be changed to obtain packing B'. □

In order to prove correctness of Algorithm 3.7, we will make use of the auxiliary Algorithm 3.14 (REDUCECOMPONENTS). Due to a delete-operation, the value of the optimal solution $\operatorname{OPT}(I, s)$ might decrease. Since the number of non-zero components has to be bounded by $C = \Delta\operatorname{OPT}(I, s) + m$, the number of non-zero components might have to be adjusted down. The following algorithm describes how a fractional solution x' and an integral solution y' with reduced number of non-zero components can be computed such that $\|y - y'\|_1$ is bounded. The idea behind the algorithm is also used in the IMPROVE algorithm. The smallest $m+2$ components are reduced to $m+1$ components using a standard technique presented for example in [BM98]. Arbitrary many components of x' can thus be reduced to $m+1$ components without making the approximation guarantee worse.

Algorithm 3.14 (REDUCECOMPONENTS)**.**

1. *Choose the smallest non-zero components* $y_{a_1}, \ldots, y_{a_{m+2}}$.
2. *If* $\sum_{1 \leq i \leq m+2} y_{a_i} \geq (1/\Delta + 2)(m+2)$ *then return* $x = x'$ *and* $y = y'$
3. *Reduce the components* $x_{a_1}, \ldots, x_{a_{m+2}}$ *to* $m+1$ *components* $\hat{x}_{b_1}, \ldots, \hat{x}_{b_{m+1}}$ *with* $\sum_{j=1}^{m+2} x_{a_j} = \sum_{j=1}^{m+1} \hat{x}_{b_j}$.
4. *For all* i *set* $x'_i = \begin{cases} \hat{x}_i + x_i & \text{if } i = b_j \text{ for some } j \leq m \\ 0 & \text{if } i = a_j \text{ for some } j \leq m+1 \\ x_i & \text{else} \end{cases}$

$$\text{and } \hat{y}_i = \begin{cases} \lceil \hat{x}_i + x'_i \rceil & \text{if } i = b_j \text{ for some } j \leq m \\ 0 & \text{if } i = a_j \text{ for some } j \leq m+1 \\ y_i & \text{else} \end{cases}$$

5. *If possible choose $d \in V(\hat{y} - x')$ such that $\|d\|_1 = m+1$ otherwise choose $d \in V(\hat{y} - x')$ such that $\|d\|_1 < m+1$ is maximal.*

6. *Return $y' = \hat{y} - d$*

The following theorem shows that the algorithm above yields a new fractional solution x' and a new integral solution y' with a reduced number of non-zero components.

Theorem 3.15. *Let x be a fractional solution of the LP with $\|x\|_1 \leq (1+\Delta)\,\mathrm{OPT}(I,s)$. Let y be an integral solution of the LP with $\|y\|_1 \leq (1+2\Delta)\,\mathrm{OPT}(I,s) + m$. Suppose x and y have the same number $\leq C+1$ of non-zero components and for all components i we have $y_i \geq x_i$. Using the Algorithm* ReduceComponents *on x and y returns a new fractional solution x' with $\|x'\|_1 \leq (1+\Delta)\,\mathrm{OPT}(I,s)$ and a new integral solution y' with $\|y'\|_1 \leq (1+2\Delta)\,\mathrm{OPT}(I,s) + m$. Further, both solutions x' and y' have the same number of non-zero components and for each component we have $x'_i \leq y'_i$. The number of non-zero components can now be bounded by $\leq C$. Furthermore, we have that $\|y - y'\|_1 \leq 2 \cdot (1/\Delta + 3)(m+2)$.*

Proof. Case 1: $\sum_{1 \leq i \leq m+2} y_{a_i} \geq (1/\Delta + 2)(m+2)$. We will show that in this case, x and y already have $\leq C$ non-zero components. In this case the algorithm returns $x' = x$ and $y' = y$. Since $\sum_{1 \leq i \leq m+2} y_{a_i} \geq (1/\Delta + 2)(m+2)$ the components $y_{a_1}, \ldots, y_{a_{m+2}}$ have an average size of at least $(1/\Delta + 2)$ and since $y_{a_1}, \ldots, y_{a_{m+2}}$ are the smallest components, all components of y have average size at least $(1/\Delta + 2)$. The size $\|y\|_1$ is bounded by $(1+2\Delta)\,\mathrm{OPT}(I,s) + m$. Hence the number of non-zero components can be bounded by $\frac{(1+2\Delta)\,\mathrm{OPT}(I,s)+m}{1/\Delta+2} \leq \Delta\,\mathrm{OPT}(I,s) + \Delta m \leq C$.

Case 2: $\sum_{1 \leq i \leq m+1} y_{a_i} < (1/\Delta + 2)(m+2)$. We have to prove different properties for the new fractional solution x' and the new integral solution y'.

Number of non-zero components: The only change in the number of non-zero components is in step 3 of the algorithm, where the number of non-zero components is reduced by 1. As x, y have at most $C+1$ non-zero components, x', y' have at most C non-zero components. In step 4 of the algorithm, $\hat{y}$ is defined such that $\hat{y}_i \geq x'_i$. In step 5 of the algorithm d is chosen such that $\hat{y}_i - d \geq x'_i$. Hence we obtain that $y'_i = \hat{y}_i - d \geq x'_i$.

Distance between y and y': The only steps where components of y changes are in step 4 and 5. The distance between y and $\hat{y}$ is bounded by the sum of the components that are set to 0, i. e., $\sum_{j=1}^{m+2} y_{a_j}$ and the sum of the increase of the increased components $\sum_{j=1}^{m+1} \lceil \hat{x}_{b_j} \rceil \leq \sum_{j=1}^{m+1} \hat{x}_{b_j} + m + 1 = \sum_{j=1}^{m+2} x_{a_j} + m + 1$. As $\sum_{j=1}^{m+2} x_{a_j} \leq \sum_{j=1}^{m+2} y_{a_j} < (1/\Delta + 2)(m+2)$, we obtain that the distance between y and $\hat{y}$ is bounded by $2 \cdot (1/\Delta + 2)(m+2) + m + 1$. Using that $\|d\|_1 \leq m+1$, the distance between y and y' is bounded by $\|y' - y\|_1 < 2 \cdot (1/\Delta + 3)(m+2)$.

Approximation guarantee: The fractional solution x is modified by condition of step 3 such that the sum of the components does not change. Hence $\|x'\|_1 = \|x\|_1 \leq (1+\Delta)\,\mathrm{OPT}(I,s)$.

Case 2a: $\|d\|_1 < m+1$. Since d is chosen maximally we have for every non-zero component that $y'_i - x'_i < 1$. Since there are at most $C = \Delta\,\mathrm{OPT}(I,s) + m$ non-zero components we obtain that $\|y'\|_1 \leq \|x'\|_1 + C \leq (1+2\Delta)\,\mathrm{OPT}(I,s) + m$. Case 2b: $\|d\|_1 = m+1$. By

definition of $\hat{y}$ we have $\|\hat{y}\|_1 \leq \|y\|_1 + \sum_{j=1}^{m+1} \lceil \hat{x_{b_j}} + x_{b_j} \rceil - \sum_{j=1}^{m+2} x_{a_j} \leq \|y\|_1 + m + 1$. We obtain for y' that $\|y'\|_1 = \|\hat{y}\|_1 - \|d\|_1 \leq \|y\|_1 + m + 1 - (m+1) = \|y\|_1 \leq (1+2\Delta)\,\text{OPT}(I,s) + m$. □

Theorem 3.16. *Algorithm 3.7 is an AFPTAS with migration factor at most* $\mathcal{O}(\frac{1}{\epsilon^3} \cdot \log(1/\epsilon))$ *for the fully dynamic bin packing problem with respect to large items.*

Proof. Set $\delta = \epsilon$. Then $\Delta = 2\epsilon + \epsilon^2 = \mathcal{O}(\epsilon)$. We assume in the following that $\Delta \leq 1$ (which holds for $\epsilon \leq \sqrt{2} - 1$).

We prove by induction that four properties hold for any packing B_t and the corresponding LP solutions. Let x be a fractional solution of the LP defined by the instance (I_t, s^{R_t}) and y be an integral solution of this LP. The properties (2) to (4) are necessary to apply Theorem 3.13 and property (1) provides the wished approximation ratio for the bin packing problem.

(1) $\max_i B_t(i) = \|y\|_1 \leq (1 + 2\Delta)\,\text{OPT}(I(t), s) + m$ (the number of bins is bounded)

(2) $\|x\|_1 \leq (1 + \Delta)\,\text{OPT}(I(t), s)$

(3) for every configuration i we have $x_i \leq y_i$

(4) x and y have the same number of non-zero components and that number is bounded by $\Delta\,\text{OPT}(I(t), s) + m$

To apply Theorem 3.13 we furthermore need a guaranteed minimal size for $\|x\|_1$ and $\|y\|_1$. According to Theorem 3.13 the integral solution y needs $\|y\|_1 \geq (m+2)(1/\delta + 2)$ and $\|x\|_1 \geq 8(1/\delta + 1)$ as we set $\alpha \leq 4$. By condition of the while-loop the call of IMPROVE is made iff $SIZE(I_t, s) \geq 8(1/\delta + 1)$ and $SIZE(I_t, s) \geq (m+2)(1/\delta + 2)$. Since $\|y\|_1 \geq \|x\|_1 \geq SIZE(I_t, s)$ the requirements for the minimum size are fulfilled. As long as the instance is smaller than $8(1/\delta + 1)$ or $(m+2)(1/\delta + 2)$ an offline algorithm for bin packing is used. Note that we can use the offline algorithm 2.18 to fulfill properties (1) to (4) as shown in chapter 2.

Now let B_t be a packing with $SIZE(I_t, s) \geq 8(1/\delta + 1)$ and $SIZE(I_t, s) \geq (m+2)(1/\delta + 2)$ for instance I_t with solutions x and y of the LP defined by $(I(t), s^{R_t})$. Suppose by induction that the properties (1) to (4) hold for the instance I_t. We have to prove that these properties also hold for the instance $I(t+1)$ and the corresponding solutions x'' and y''. The packing B_{t+1} is created by the repeated use of an call of IMPROVE for x and y followed by an operation (insert, delete, shiftA or shiftB). We will prove that the properties (1) to (4) hold after a call of IMPROVE followed by an operation.

improve: Let x' be the resulting fractional solution of Theorem 3.13, let y' be the resulting integral solution of Theorem 3.13 and let B'_t be the corresponding packing. Properties (1) to (4) are fulfilled for x, y and B_t by induction hypothesis. Hence all conditions are fulfilled to use Theorem 3.13. By Theorem 3.13 the properties (1) to (4) are still fulfilled for x', y' and B'_t and moreover we get $\|x'\|_1 \leq (1 + \Delta)\,\text{OPT}(I(t), s) - \alpha$ and $\|y'\|_1 = \max_i B'_t(i) \leq (1 + 2\Delta)\,\text{OPT}(I(t), s) + m - \alpha$ for chosen parameter α. Let x'' and y'' be the fractional and integral solution after an operation is applied to x' and y'. We have to prove that the properties (1) to (4) are also fulfilled for x'' and y''.

operations: First we take a look at how the operations modify $\|x'\|_1$ and $\|y'\|_1 = \max_i B'_t(i)$. By construction of the insertion operation, $\|x'\|_1$ and $\|y'\|$ are increased at most by 2. By construction of the delete operation, $\|x'\|_1$ and $\|y'\|_1$ are increased by 1. By construction of the shiftA and shiftB operation, $\|x'\|_1$ and $\|y'\|_1$ are increased by 1. An IMPROVE(2) call followed by an insertion operation therefore yields $\|y''\| =$

$\|y'\|_1 + 2 = (1 + 2\Delta)\,\text{OPT}(I(t), s) + m - 2 + 2 = (1 + 2\Delta)\,\text{OPT}(I(t + 1), s) + m$ since $\text{OPT}(I(t), s) \leq \text{OPT}(I(t + 1), s)$. An IMPROVE(4) call followed by a delete operation yields $\|y''\| = \|y'\|_1 + 1 = (1+2\Delta)\,\text{OPT}(I(t), s)+m-3 \leq (1+2\Delta)\,\text{OPT}(I(t+1), s)+(1+2\Delta)+m-3 \leq (1+2\Delta)\,\text{OPT}(I(t+1), s)$ since $\text{OPT}(I(t), s) \leq \text{OPT}(I(t+1), s)+1$ (an item is removed) and $\Delta \leq 1$. In the same way we obtain that $\|y''\|_1 \leq \|y'\|_1 + 1 \leq (1+2\Delta)\,\text{OPT}(I(t+1), s)+m$ for an IMPROVE(1)/IMPROVE(3) call followed by a shiftA/shiftB operation. This concludes the proof that property (1) is fulfilled for $I(t + 1)$. The proof that property (2) holds is analog since $\|x'\|_1$ increases in the same way as $\|y'\|_1$ and $\|x'\|_1 \leq (1+\Delta)\,\text{OPT}(I(t), s) - \alpha$. For property (3) note that in the operations a configuration x_i of the fractional solution is increased by 1 if and only if a configuration y_i is increased by 1. Therefore the property that for all configurations $x''_i \leq y''_i$ retains from x' and y'. By Theorem 3.13 the number of non-zero components of x' and y' is bounded by $\Delta\,\text{OPT}(I(t), s)+m \leq \Delta\,\text{OPT}(I(t+1), s)+m$ in case of an insert operation. If an item is removed, the number of non-zero components of x' and y' is bounded by $\Delta\,\text{OPT}(I(t), s) + m \leq \Delta\,\text{OPT}(I(t + 1), s) + m + 1 = C + 1$. By Theorem 3.15 the algorithm REDUCECOMPONENTS guarantees that there are at most $C = \Delta\,\text{OPT}(I(t + 1), s) + m$ non-zero components. By construction of the shift-operation, x'' and y'' might have two additional non-zero components. But since these are being reduced by Algorithm 3.7 (note that we increased the number of components being reduced in step 6 by 2 to- see chapter 2 for details), the LP solutions x'' and y'' have at most $\Delta\,\text{OPT}(I(t + 1), s) + m$ non-zero components which proves property (4). Algorithm 3.7 therefore has an asymptotic approximation ratio of $1 + \epsilon$.

We still need to examine the migration factor of Algorithm 3.7. In the case that the offline algorithm is used, the size of the instance is smaller than $8(1/\delta + 1) = \mathcal{O}(1/\epsilon)$ or smaller than $(m + 2)(1/\delta + 2) = \mathcal{O}(\frac{1}{\epsilon^2}\log(1/\epsilon))$. Hence the migration factor in that case is bounded by $\mathcal{O}(\frac{1}{\epsilon^3}\log(1/\epsilon))$. If the instance is bigger the call of IMPROVE repacks at most $\mathcal{O}(m/\epsilon)$ bins by Theorem 3.13. Since every large arriving item has size $> \epsilon/14$ and $m = \mathcal{O}(\frac{1}{\epsilon}\log(1/\epsilon))$ we obtain a migration factor of $\mathcal{O}(\frac{1}{\epsilon^3}\log(1/\epsilon))$ for the Algorithm IMPROVE. Since the migration factor of each operation is also bounded by $\mathcal{O}(\frac{1}{\epsilon^2}\log(1/\epsilon))$, we obtain an overall migration factor of $\mathcal{O}(\frac{1}{\epsilon^3}\log(1/\epsilon))$.

The main complexity of Algorithm 3.7 lies in the use of Algorithm IMPROVE. As described in chapter 2 the running time of IMPROVE is bounded by $\mathcal{O}(M(1/\epsilon\log(1/\epsilon)) \cdot 1/\epsilon^3\log(1/\epsilon))$, where $M(n)$ is the time needed to solve a system of n linear equations. By using heap structures to store the items, each operation can be performed in time $\mathcal{O}(1/\epsilon\log(1/\epsilon) \cdot \log(\epsilon^2 \cdot n(t)))$ at time t, where $n(t)$ denotes the number of items in the instance at time t. As the number of non-zero components is bounded by $\mathcal{O}(\epsilon \cdot n(t))$, the total running time of the algorithm is bounded by $\mathcal{O}(M(1/\epsilon\log(1/\epsilon)) \cdot 1/\epsilon^3\log(1/\epsilon) + 1/\epsilon\log(1/\epsilon)\log(\epsilon^2 \cdot n(t)) + \epsilon n(t))$. The best known running time for the dynamic bin packing problem *without* removals was $\mathcal{O}(M(1/\epsilon^2) \cdot 1/\epsilon^4 + \epsilon n(t) + \frac{1}{\epsilon^2}\log(\epsilon^2 n(t)))$ as presented in chapter 2. As this is polynomial in $n(t)$ and in $1/\epsilon$ we can conclude that Algorithm 3.7 is an AFPTAS. □

If no deletions are present, we can use a simple FirstFit algorithm (as described by in chapter 2) to pack the small items into the bins. This does not change the migration factor or the running time of the algorithm and we obtain a robust AFPTAS with $\mathcal{O}(\frac{1}{\epsilon^3} \cdot \log(1/\epsilon))$ migration for the case that no items is removed. This improves upon the best known migration factor of $\mathcal{O}(\frac{1}{\epsilon^4})$ that we presented in chapter 2 and which was published in [JK13].

3.4 Handling Small Items

In this section we present methods for dealing with arbitrary small items in a dynamic online setting. First, we present a robust AFPTAS with migration factor of $\mathcal{O}(1/\epsilon)$ for the case that only small items arrive and depart. In Section 3.4.3 we generalize these techniques to a setting where small items arrive into a packing where large items are already packed and can not be rearranged. Finally we state the AFPTAS for the general fully dynamic bin packing problem. In a robust setting without departing items, small items can easily be treated by packing them greedily via the classical FirstFit algorithm of Johnson et al. [Joh+74b] (see Epstein and Levin [EL09] or chapter 2 of this thesis). However, in a setting where items may also depart, small items need to be treated much more carefully. We show that the FirstFit algorithm does not work in this dynamic setting.

Lemma 3.17. *Using the FirstFit algorithm to pack small items may lead to an arbitrarily bad approximation.*

Proof. Suppose, that there is an algorithm $\mathcal{A}$ with migration factor c which uses FirstFit on items with size $< \epsilon/14$. We will now construct an instance where $\mathcal{A}$ yields an arbitrary bad approximation ratio. Let $b = \epsilon/14 - \delta$ and $a = \epsilon/14c - (\delta + c\delta)/c$ for a small δ such that $(1-b)/a$ is integral. Note that $ac < b$ by definition. Furthermore, let $M \in \mathbb{N}$ be an arbitrary integer and consider the instance

$$I_M = [\underbrace{A, A, \ldots, A}_{M}, \underbrace{B, B, \ldots, B}_{M}]$$

with

$$A = (b, \text{Insert}), \underbrace{(a, \text{Insert}), (a, \text{Insert}), \ldots, (a, \text{Insert})}_{(1-b)/a}$$

$$B = \underbrace{(a, \text{Delete}), (a, \text{Delete}), \ldots, (a, \text{Delete})}_{(1-b)/a}.$$

After the insertion of all items, there are M bins containing an item of size b and $1-b/a$ items of size a (see Figure 3.8a). As $ac < b$, the deletion of the items of size a can not move the items of size b. The remaining M bins thus only contain a single item of size b (see Figure 3.8b), while $\lceil M \cdot b \rceil$ bins would be sufficient to pack all of the remaining items. The approximation ratio is thus at least $M/M \cdot b = 1/b \approx \frac{1}{\epsilon}$ and thus grows as ϵ shrinks. In order to avoid this problem, we design an algorithm which groups items of similar size together. Using such a mechanism would therefore put the second item of size b into the first bin by shifting out an appropriate number of items of size a and so on. Our algorithms achieves this grouping of small items by enumerating the bins and maintaining the property, that larger small items are always left of smaller small items.

□

3.4.1 Only Small Items

We consider a setting where only small items exist, i. e., items with a size less than $\epsilon/14$. First, we divide the set of small items into different size intervals S_j where $S_j = \left[\frac{\epsilon}{2^{j+1}}, \frac{\epsilon}{2^j}\right)$ for $j \geq 1$. Let $b_1, \ldots, b_m$ be the used bins of our packing. We say a size category S_j is

(a) A single bin after the insertion (b) A single bin after the deletion

Figure 3.8: Construction in the proof of Lemma 3.17

bigger than a size category S_k if $j < k$, i. e., the item sizes contained in S_j are larger (note that a size category S_j with large index j is called small). We say a bin b_i is filled completely if it has less than $\frac{\epsilon}{2^j}$ remaining space, where S_j is the biggest size category appearing in b_i. Furthermore we label bins b_i as *normal* or as *buffer bins* and partition all bins $b_1, \ldots, b_m$ into *queues* $Q_1, \ldots, Q_d$ for $|Q| \leq m$. A queue is a subsequence of bins $b_i, b_{i+1} \ldots, b_{i+c}$ where bins $b_i, \ldots, b_{i+c-1}$ are normal bins and bin b_{i+c} is a buffer bin. We denote the i-th queue by Q_i and the number of bins in Q_i by $|Q_i|$. The buffer bin of queue Q_i is denoted by bb_i.

We will maintain a special form for the packing of small items such that the following properties are always fulfilled. For the sake of simplicity, we assume that $1/\epsilon$ is integral.

(1) For every item $i \in b_d$ with size $s(i) \in S_j$ for some $j, d \in \mathbb{N}$, there is no item $i' \in b_{d'}$ with size $s(i') \in S_{j'}$ such that $d' > d$ and $j' > j$. This means: Items are ordered from left to right by their size intervals.

(2) Every normal bin is filled completely.

(3) The length of each queue is at least $1/\epsilon$ and at most $2/\epsilon$ except for the last queue Q_d.

Note that property (1) implies that all items in the same size interval S_j are packed into bins $b_x, b_{x+1}, \ldots, b_{x+c}$ for constants x and c. Items in the next smaller size category S_{j+1} are then packed into bins $b_{x+c}, b_{x+c+1}, \ldots$ and so on. We denote by $b_{S(\ell)}$ the last bin in which an item of size interval S_ℓ appears. We denote by $S_{>\ell}$ the set of smaller size categories $S_{\ell'}$ with $\ell' > \ell$. Note that items in size category $S_{>\ell}$ are smaller than items in size category S_ℓ.

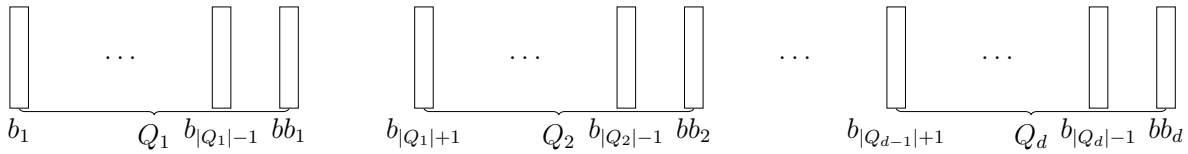


Figure 3.9: Distribution of bins with small items into queues

The following lemma guarantees that a packing that fulfills properties (1) to (3) is close to the optimum solution.

Lemma 3.18. *If properties* (1) *to* (3) *hold, then at most* $(1 + \mathcal{O}(\epsilon))\,\mathrm{OPT}(I, s) + 2$ *bins are used in the packing for every* $\epsilon \leq 1/3$.

Proof. Let C be the number of used bins in our packing. By property (2) we know that all normal bins have less than $\epsilon/14$ free space. Property (3) implies that there are at most $\epsilon \cdot C + 1$ buffer bins and hence possibly empty. The number of normal bins is thus at least $(1 - \epsilon) \cdot C - 1$. Therefore we can bound the total size of all items by

$\geq (1 - \epsilon/14) \cdot ((1-\epsilon) \cdot C - 1)$. As $\text{OPT}(I,s) \geq SIZE(I,s) \geq (1-\epsilon/14) \cdot ((1-\epsilon) \cdot C - 1)$ and $\frac{1}{(1-\epsilon/14)(1-\epsilon)} \leq 1 + 2\epsilon$ for $\epsilon \leq 1/3$ we get $C \leq (1+2\epsilon)\,\text{OPT}(I,s) + 2$. □

We will now describe the operations that are applied whenever a small item has to be inserted or removed from the packing. The operations are designed such that properties (1) to (3) are never violated and hence a good approximation ratio can be guaranteed by Lemma 3.18 at every step of the algorithm. The operations are applied recursively such that some items from each size interval are shifted from left to right (insert) or right to left (delete). The recursion halts if the first buffer bin is reached. Therefore, the free space in the buffer bins will change over time. Since the recursion always halts at the buffer bin, the algorithm is applied on a single queue Q_k.

The following Insert/Delete operation is defined for a whole set $J = \{i_1, \ldots, i_n\}$ of items. If an item i of size interval S_ℓ has to be inserted or deleted, the algorithm is called with Insert$(\{i\}, b_{S(\ell)}, Q_k)$ respectively Delete$(\{i\}, b_x, Q_k)$, where b_x is the bin containing item i and Q_k is the queue containing bin $b_{S(\ell)}$ or b_x. Recall that $S_j = \left[\frac{\epsilon}{2^{j+1}}, \frac{\epsilon}{2^j}\right)$ is a fixed interval for every $j \geq 1$ and $S_{\leq j} = \bigcup_{i=1}^{j} S_i$ and $S_{>j} = \bigcup_{i>j} S_i$.

Algorithm 3.19 (Insert or Delete for only small items)**.**

- **Insert**(J, b_x, Q_k)**:**
  - *Insert the set of small items $J = \{i_1, \ldots, i_n\}$ with size $s(i_j) \in S_{\leq \ell}$ into bin b_x. (By Lemma 3.21 the total size of J is bounded by $\mathcal{O}(1/\epsilon)$ times the size of the item which triggered the first Insert operation.)*
  - *Remove just as many items $J' = \{i'_1, \ldots, i'_m\}$ of the smaller size interval $S_{>\ell}$ appearing in bin b_x (starting by the smallest) such that the items $i_1, \ldots, i_n$ fit into the bin b_x. If there are not enough items of smaller size categories to insert all items from I, insert the remaining items from I into bin b_{x+1}.*
  - *Let $J'_{\ell'} \subseteq J'$ be the items in the respective size interval $S_{\ell'}$ with $\ell' > \ell$. Put the items $J'_{\ell'}$ recursively into bin $b_{S(\ell')}$ (i. e., call Insert$(J'_{\ell'}, b_{S(\ell')}, Q_k)$ for each $\ell' > \ell$). If the buffer bin bb_k is left of $b_{S(\ell')}$ call Insert$(J'_{\ell'}, bb_k, Q_k)$ instead.*
- **Delete**(J, b_x, Q_k)**:**
  - *Remove the set of items $J = \{i_1, \ldots, i_n\}$ with size $s(i_j) \in S_{\leq \ell}$ from bin b_x (By Lemma 3.21 the total size of J is bounded by $\mathcal{O}(1/\epsilon)$ times the size of the item which triggered the first Delete operation.)*
  - *Insert as many small items $J' = \{i'_1, \ldots, i'_m\}$ from $b_{S(\ell')}$, where $S_{\ell'}$ is the smallest size interval appearing in b_x such that b_x is filled completely. If there are not enough items from the size category $S_{\ell'}$, choose items from size category $S_{\geq \ell'+1}$ in bin b_{x+1}.*
  - *Let $J'_{\ell'} \subseteq J'$ be the items in the respective size interval $S_{\ell'}$ with $\ell' > \ell$. Remove items $J'_{\ell'}$ from bin $b_{S(\ell')}$ recursively (i. e., call Delete$(J'_{\ell'}, b_{S(\ell')}, Q_k)$ for each $\ell' > \ell$). If the buffer bin bb_k is left of $b_{S(\ell')}$, call Delete$(J'_{\ell'}, bb_k, Q_k)$ instead.*

Using the above operations maintains the property of normal bins to be filled completely. However, the size of items in buffer bins changes. In the following we describe how to handle buffer bins that are being emptied or filled completely.

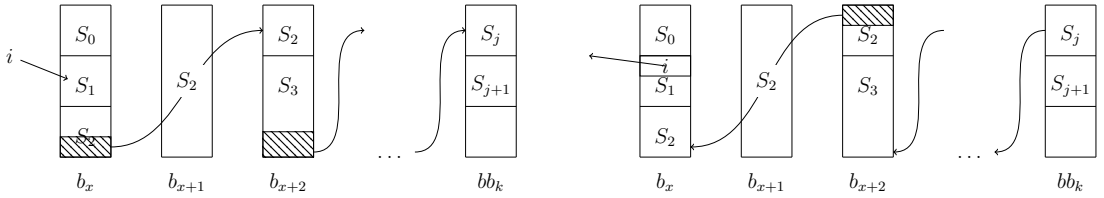
Algorithm 3.20 (Handle filled or emptied buffer bins)**.**

- **Case 1: The buffer bin of Q_i is filled completely by an insert operation.**
  - *Label the filled bin as a normal bin and add a new empty buffer bin to the end of Q_i.*

- *If $|Q_i| > 2/\epsilon$, split Q_i into two new queues Q'_i, Q''_i with $|Q''_i| = |Q'_i| + 1$. The buffer bin of Q''_i is the newly added buffer bin. Add an empty bin labeled as the buffer bin to Q'_i such that $|Q'_i| = |Q''_i|$.*

- **Case 2: The buffer bin of Q_i is being emptied due to a delete operation.**
  - *Remove the now empty bin.*
  - *If $|Q_i| \geq |Q_{i+1}|$ and $|Q_i| > 1/\epsilon$, choose the last bin of Q_i and label it as new buffer bin of Q_i.*
  - *If $|Q_{i+1}| > |Q_i|$ and $|Q_{i+1}| > 1/\epsilon$, choose the first bin of Q_{i+1} and move the bin to Q_i and label it as buffer bin.*
  - *If $|Q_{i+1}| = |Q_i| = 1/\epsilon$, merge the two queues Q_i and Q_{i+1}. As Q_{i+1} already contains a buffer bin, there is no need to label another bin as buffer bin for the merged queue.*

Creating and deleting buffer bins this way guarantees that property (3) is never violated since queues never exceed the length of $2/\epsilon$ and never fall below $1/\epsilon$.



(a) Insert($\{i\}$,b_x,Q_k) with $s(i) \in S_1$ (b) Delete($\{i\}$,b_x,Q_k) with $s(i) \in S_1$

Figure 3.10: Example calls of Insert and Delete.

Figure 3.10a shows an example call of Insert($\{i\}$,b_x,Q_k). Item i with $s(i) \in S_1$ is put into the corresponding bin b_x into the size interval S_1. As b_x now contains too many items, some items from the smallest size interval S_2 (marked by the dashed lines) are put into the last bin b_{x+2} containing items from S_2. Those items in turn push items from the smallest size interval S_3 into the last bin containing items of this size and so on. This process terminates if either no items need to be shifted to the next bin or the buffer bin bb_k is reached.

It remains to prove that the migration of the operations is bounded and that the properties are invariant under those operations.

Lemma 3.21.

(i) Let I be an instance that fulfills properties (1) *to* (3)*. Applying operations insert/delete on I yields an instance I' that also fulfills properties* (1) *to* (3)*.*

(ii) The migration factor of a single insert/delete operation is bounded by $\mathcal{O}(1/\epsilon)$ for all $\epsilon \leq 2/7$.

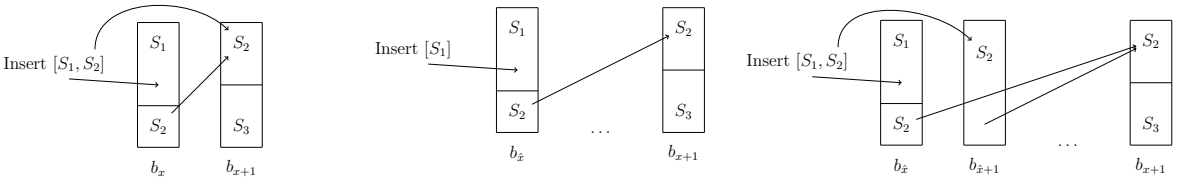
Proof. Proof for (i): Suppose the insert/delete operation is applied to a packing which fulfills properties (1) to (3). By construction of the insert operation, items from a size category S_ℓ in bin b_x are shifted to a bin b_y. The bin b_y is either $b_{S(\ell)}$ or the a buffer bin left of $b_{S(\ell)}$. By definition b_y contains items of size category S_ℓ. Therefore property (1) is not violated. Symmetrically, by construction of the delete operation, items from a size category S_ℓ in bin $b_{S(\ell)}$ are shifted to a bin b_x. By definition b_x contains items of size category S_ℓ and property (1) is therefore not violated. For property (2): Let b_x be

a normal bin, where items $i_1, \ldots, i_n$ of size category $S_{\leq \ell}$ are inserted. We have to prove that the free space in b_x remains smaller than $\epsilon/2^j$, where S_j is the smallest size category appearing in bin b_x. By construction of the insert operation, just as many items of size categories $S_{>\ell}$ are shifted out of bin b_x such that $i_1, \ldots, i_n$ fit into b_x. Hence the remaining free space is less than $\frac{\epsilon}{2^\ell}$ and bin b_x is filled completely. The same argumentation holds for the delete operation. Property (3) is always fulfilled by definition of Algorithm 3.20.

Proof for (ii): According to the insert operation, in every recursion step of the algorithm, it tries to insert a set of items into a bin $b_{x'}$, starting with an Insert$(\{i\}, b_{x'}, Q_k)$ operation. Let $\text{INSERT}(S_{\leq \ell+y}, b_x)$ $(x \geq x')$ be the size of all items in size categories S_j with $j \leq \ell + y$ that the algorithm tries to insert into b_x as a result of an Insert$(\{i\}, b_{x'}, Q_k)$ call. Let $\text{PACK}(b_x)$ be the size of items that are actually packed into bin b_x. We have to distinguish between two cases. In the case that $\text{INSERT}(S_{\leq \ell+y}, b_x) = \text{PACK}(b_x)$ there are enough items of smaller size categories $S_{>\ell+y}$ that can be shifted out, such that items I fit into bin b_x. In the case that $\text{INSERT}(S_{\leq \ell+y}, b_x) > \text{PACK}(b_x)$ there are not enough items of smaller size category that can be shifted out and the remaining size of $\text{INSERT}(S_{\leq \ell+y}, b_x) - \text{PACK}(b_x)$ has to be shifted to the following bin b_{x+1}. Under the assumption that each $\text{INSERT}(S_{\leq \ell}, b_x) \leq 1$ for all x and ℓ (which is shown in the following) all items fit into b_{x+1}. Note that no items from bins left of b_x can be shifted into b_{x+1} since $b_x = b_{S(\ell+y)}$ is the last bin where items of size category $S_{\leq \ell+y}$ appear. Hence all items shifted out from bins left of b_x are of size categories $S_{\leq \ell+y}$ (property (1)) and they are inserted into bins left of b_{x+1}. We prove by induction that for each $\text{INSERT}(S_{\leq \ell+y}, b_x)$ the total size of moved items is at most

$$\text{INSERT}(S_{\leq \ell+y}, b_x) \leq s(i) + 3\sum_{j=1}^{y} \frac{\epsilon}{2^{\ell+j}}$$

The claim holds obviously for $\text{INSERT}(S_{\leq \ell}, b_{x'})$ since $b_{x'} = b_{S(\ell)}$ is the bin where only item i is inserted.



(a) Case 1 (b) Case 2a (c) Case 2b

Figure 3.11: All cases to consider in Lemma 3.21

Case 1: $\text{INSERT}(S_{\leq \ell+y}, b_x) > \text{PACK}(b_x)$
In this case, the size of all items that have to be inserted into b_{x+1} can be bounded by the size of items that did not fit into bin b_x plus the size of items that were removed from bin b_x. We can bound $\text{INSERT}(S_{\leq \ell+\bar{y}}, b_{x+1})$ where $\bar{y} > y$ is the largest index $S_{\ell+\bar{y}}$ appearing in bin b_x by

$$\text{INSERT}(S_{\leq \ell+y}, b_x) + \frac{\epsilon}{2^{\ell+y}} \leq s(i) + 3\sum_{j=1}^{y} \frac{\epsilon}{2^{\ell+j}} + 2\frac{\epsilon}{2^{\ell+y+1}} < s(i) + 3\sum_{j=1}^{y+1} \frac{\epsilon}{2^{\ell+j}}$$

Case 2: $\text{INSERT}(S_{\leq \ell+y}, b_x) = \text{PACK}(b_x)$
Suppose that the algorithm tries to insert a set of items I of size categories $S_{\leq \ell+\bar{y}}$ into the bin $b_{x+1} = b_{S(\ell+\bar{y})}$. The items I can only be shifted from previous bins where items

of size category $S_{\leq \ell+\bar{y}}$ appear. There are only two possibilities remaining. Either all items I are shifted from a single bin $b_{\hat{x}}$ ($\hat{x} \leq x$) or from two consecutive bins $b_{\hat{x}}, b_{\hat{x}+1}$ with $\text{INSERT}(S_{\leq \ell+y}, b_{\hat{x}}) > \text{PACK}(b_{\hat{x}})$.

Note that b_{x+1} can only receive items from more than one bin if there are two bins $b_{\hat{x}}, b_{\hat{x}+1}$ with $\text{INSERT}(S_{\leq \ell+y}, b_{\hat{x}}) > \text{PACK}(b_{\hat{x}})$ such that $b_{x+1} = b_{S(\ell+\bar{y})}$ and all items shifted out of $b_{\hat{x}}, b_{\hat{x}+1}$ and into b_{x+1} are of size category $S_{\ell+\bar{y}}$. Hence bins left of $b_{\hat{x}}$ or right of $b_{\hat{x}+1}$ can not shift items into b_{x+1}.

Case 2a: All items I are shifted from a single bin $b_{\hat{x}}$ with $\hat{x} \leq x$ (note that $\hat{x} < x$ is possible since $\text{PACK}(b_x) = \text{INSERT}(S_{\leq \ell+y}, b_x)$ can be zero). The total size of items that are shifted out of $b_{\hat{x}}$ can be bounded by $\text{INSERT}(S_{\leq \ell+y}, b_{\hat{x}}) + \frac{\epsilon}{2^{\ell+y}}$. By induction hypothesis $\text{INSERT}(S_{\leq \ell+y}, b_{\hat{x}})$ is bounded by $s(i) + 3\sum_{j=1}^{y} \frac{\epsilon}{2^{\ell+j}}$. Since all items that are inserted into b_{x+1} come from $b_{\hat{x}}$, the value $\text{INSERT}(S_{\leq \ell+\bar{y}}, b_{x+1})$ ($\bar{y} > y$) can be bounded by $\text{INSERT}(S_{\leq \ell+y}, b_{\hat{x}}) + \frac{\epsilon}{2^{\ell+y}} \leq s(i) + 3\sum_{j=1}^{y} \frac{\epsilon}{2^{\ell+j}} + \frac{\epsilon}{2^{\ell+y}} < s(i) + 3\sum_{j=1}^{\bar{y}} \frac{\epsilon}{2^{\ell+j}}$ where $S_{\ell+\bar{y}}$ is the smallest size category inserted into b_{x+1}. Note that the items I belong to only one size category $S_{\ell+\bar{y}}$ if $\hat{x} < x$ since all items that are in size intervals $S_{<\ell+\bar{y}}$ are inserted into bin $b_{\hat{x}+1}$.

Case 2b: Items I are shifted from bins $b_{\hat{x}}$ and $b_{\hat{x}+1}$ ($\hat{x}+1 \leq x$) with $\text{INSERT}(S_{\leq \ell+y}, b_{\hat{x}}) > \text{PACK}(b_{\hat{x}})$. In this case, all items I belong to the size category $S_{\ell+\bar{y}}$ since $b_{\hat{x}}$ is left of b_x. Hence all items which are inserted into $b_{\hat{x}+1}$ are from I, i. e., $\text{INSERT}(S_{\leq \ell+y}, b_{\hat{x}}) = \text{PACK}(b_{\hat{x}}) + \text{PACK}(b_{\hat{x}+1})$ as all items in I belong to the same size category $S_{\ell+\bar{y}}$. We can bound $\text{INSERT}(S_{\ell+\bar{y}}, b_{x+1})$ by the size of items that are shifted out of $b_{\hat{x}}$ plus the size of items that are shifted out of $b_{\hat{x}+1}$. We obtain

$$
\begin{aligned}
&\text{INSERT}(S_{\leq \ell+\bar{y}}, b_{x+1}) \leq \text{PACK}(b_{\hat{x}}) + \frac{\epsilon}{2^{\ell+y}} + \text{PACK}(b_{\hat{x}+1}) + \frac{\epsilon}{2^{\ell+\bar{y}}} \\
&= \text{INSERT}(S_{\leq \ell+y}, b_{\hat{x}})) + \frac{\epsilon}{2^{\ell+y}} + \frac{\epsilon}{2^{\ell+\bar{y}}} \\
&\leq s(i) + 3\sum_{j=1}^{y} \frac{\epsilon}{2^{\ell+j}} + \frac{\epsilon}{2^{\ell+y}} + \frac{\epsilon}{2^{\ell+\bar{y}}} \\
&\leq s(i) + 3\sum_{j=1}^{y} \frac{\epsilon}{2^{\ell+j}} + 3\frac{\epsilon}{2^{\ell+\bar{y}}} \leq s(i) + 3\sum_{j=1}^{\bar{y}} \frac{\epsilon}{2^{\ell+j}}
\end{aligned}
$$

This yields that $\text{INSERT}(S_{\leq \ell+y}, b_x)$ is bounded by $s(i) + 3\sum_{j=1}^{\bar{y}} \frac{\epsilon}{2^{\ell+j}}$ for all bins b_x in Q_k. Now, we can bound the migration factor for every bin b_x of Q_k for any $y \in \mathbb{N}$ by $\text{PACK}(b_x) + \frac{\epsilon}{2^{\ell+y}} \leq \text{INSERT}(S_{\leq \ell+y}, b_x) + \frac{\epsilon}{2^{\ell+y}}$. Using the above claim, we get:

$$
\begin{aligned}
&\text{INSERT}(S_{\leq \ell+y}, b_x) + \frac{\epsilon}{2^{\ell+y}} \leq s(i) + 3\sum_{j=1}^{y} \frac{\epsilon}{2^{\ell+j}} + 2\frac{\epsilon}{2^{\ell+y+1}} \\
&< s(i) + 3\sum_{j=1}^{\infty} \frac{\epsilon}{2^{\ell+j}} = s(i) + 3\frac{\epsilon}{2^{\ell}} \sum_{j=1}^{\infty} \frac{1}{2^{j}} = s(i) + 3 \cdot \frac{\epsilon}{2^{\ell}} \leq 7s(i)
\end{aligned}
$$

Since there are at most $2/\epsilon$ bins per queue, we can bound the total migration of the operation $\text{Insert}(\{i\}, b_{S(\ell)}, Q_k)$ by $7 \cdot 2/\epsilon \in \mathcal{O}(1/\epsilon)$. Note also that $s(i) \leq \epsilon/14$ for every i implies that $\text{INSERT}(S_{\leq \ell}, b_x)$ is bounded by $\epsilon/2$ for all x and ℓ .

Suppose that items $i_1, \ldots, i_n$ of size interval $S_{\ell+y}$ have to be removed from bin b_x. In order to fill the emerging free space, items from the same size category are moved out of $b_{S(\ell)}$ into the free space. As the bin b_x may already have additional free space, we need to move at most a size of $\text{SIZE}(i_1, \ldots, i_n) + \epsilon/2^{\ell+y}$. Using a symmetric proof as above yields a migration factor of $\mathcal{O}(\frac{1}{\epsilon})$. □

3.4.2 Handling small items in the general setting

In the scenario that there are mixed item types (small and large items), we need to be more careful in the creation and the deletion of buffer bins. To maintain the approximation guarantee, we have to make sure that as long as there are bins containing only small items, the remaining free space of all bins can be bounded. Packing small items into empty bins and leaving bins with large items untouched does not lead to a good approximation guarantee as the free space of the bins containing only large items is not used. In this section we consider the case where a sequence of small items is inserted or deleted. We assume that the packing of large items does not change. Therefore the number of bins containing large items equals a fixed constant $\Lambda(B)$. In the previous section, the bins $b_1, \ldots, b_{m(B)}$ all had a capacity of 1. In order to handle a mixed setting, we will treat a bin b_i containing large items as having capacity of $c(b_i) = 1 - S$, where S is the total size of the large items in b_i. The bins containing small items are enumerated by $b_1, \ldots, b_{L(B)}, b_{L(B)+1}, \ldots, b_{m(B)}$ for some $L(B) \leq m(B)$ where $c(b_1), \ldots, c(b_{L(B)}) < 1$ and $c(b_{L(B)+1}) = \ldots = c(b_{m(B)}) = 1$. Additionally we have a separate set of bins, called the *heap bins*, which contain only large items. This set of bins is enumerated by $h_1, \ldots h_{h(B)}$. Note that $L(B) + h(B) = \Lambda(B)$. In general we may consider only bins b_i and h_i with capacity $c(b_i) \geq \epsilon/14$ and $c(h_i) \geq \epsilon/14$ since bins with less capacity are already packed well enough for our approximation guarantee as shown by Lemma 3.21. Therefore, full bins are not considered in the following.

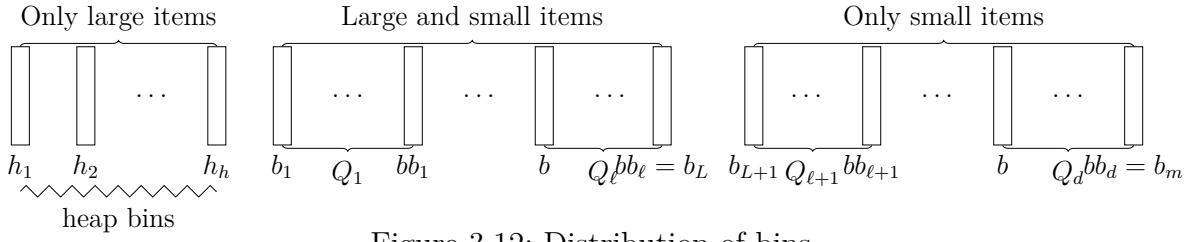


Figure 3.12: Distribution of bins

As before, we partition the bins $b_1, \ldots, b_{L(B)}, b_{L(B)+1}, \ldots, b_{m(B)}$ into several different queues $Q_1, \ldots, Q_{\ell(B)}, Q_{\ell(B)+1}, \ldots, Q_{d(B)}$ such that $b_1, \ldots b_{L(B)} = Q_1, \ldots Q_{\ell(B)}$ and $b_{L(B)+1}, \ldots b_{m(B)} = Q_{\ell(B)+1}, \ldots, Q_{d(B)}$. If the corresponding packing B is clear from the context, we will simply write $h, L, \ell, d, m, \Lambda$ instead of $h(B), L(B), \ell(B), d(B), m(B), \Lambda(B)$. We denote the last bin of queue Q_i by bb_i which is a buffer bin. The buffer bin bb_ℓ is special and will be treated differently in the insert and delete operation. Note that the bins containing large items $b_1, \ldots, b_{L(B)}$ are enumerated first. This guarantees that the free space in the bins containing large items is used before new empty bins are opened to pack the small items. However, enumerating bins containing large items first, leads to a problem if according to Algorithm 3.20 when a buffer bin is being filled and a new bin has to be inserted right to the filled bin. Instead of inserting a new empty bin, we insert a heap bin at this position. Since the heap bin contains only large items, we do not violate the order of the small items (see Figure 3.12). As the inserted heap bin has remaining free space (is not filled completely) for small items, it can be used as a buffer bin. In order to get an idea of how many heap bins we have to reserve for Algorithm 3.20 where new bins are inserted or deleted, we define a potential function. As a buffer bin is being filled or emptied completely the Algorithm 3.20 is executed and inserts or deletes buffer bins. The potential function $\Phi(B)$ thus bounds the number of buffer bins in

$Q_1, \ldots, Q_{\ell(B)}$ that are about to get filled or emptied. The potential $\Phi(B)$ is defined by

$$\Phi(B) = \sum_{i=1}^{\ell-1} r_i + \lceil \epsilon\Lambda \rceil - \ell$$

where the *fill ratio* r_i is defined by $r_i = \frac{s(bb_i)}{c(bb_i)}$ and $s(bb_i)$ is the total size of all small items in bb_i . Note that the potential only depends on the queues $Q_1, \ldots, Q_{\ell(B)}$ and the bins which contain small and large items. The term r_i intends to measure the number of buffer bins that become full. According to Case 1 of the previous section a new buffer bin is opened when bb_i is filled i. e., $r_i \approx 1$. Hence the sum $\sum_{i=1}^{\ell-1} r_i$ bounds the number of buffer bins getting filled. The term $\epsilon\Lambda$ in the potential measures the number of bins that need to be inserted due to the length of a queue exceeding $2/\epsilon$, as we need to split the queue Q_i into two queues of length $1/\epsilon$ according to Case 1. Each of those queues needs a buffer bin, hence we need to insert a new buffer bin out of the heap bins. Therefore the potential $\Phi(B)$ bounds the number of bins which will be inserted as new buffer bins according to Case 1.

Just like in the previous section we propose the following properties to bound the approximation ratio and the migration factor. The first three properties remain the same as in Section 3.4.1 and the last property gives the desired connection between the potential function and the heap bins.

(1) For every item $i \in b_d$ with size $s(i) \in S_j$ for some $j, d \in \mathbb{N}$, there is no item $i' \in b_{d'}$ with size $s(i') \in s_{j'}$ such that $d' > d$ and $j' > j$. This means: Items are ordered from left to right by their size intervals.

(2) Every normal bin of $b_1, \ldots, b_m$ is filled completely

(3) The length of each queue is at least $1/\epsilon$ and at most $2/\epsilon$ except for Q_ℓ and Q_d. The length of Q_ℓ and Q_d is only limited by $1 \leq |Q_\ell|, |Q_d| \leq 1/\epsilon$. Furthermore, $|Q_{\ell+1}| = 1$ and $1 \leq |Q_{\ell+2}| \leq 2/\epsilon$.

(4) The number of heap bins $H_1, \ldots, H_h$ is exactly $h = \lfloor \Phi(B) \rfloor$

Since bins containing large items are enumerated first, property (1) implies in this setting that bins with large items are filled before bins that contain no large items. Note also that property (3) implies that $\Phi(B) \geq 0$ for arbitrary packings B since $\epsilon\Lambda \geq \ell - 1 + \epsilon$ and thus $\lceil \epsilon\Lambda \rceil \geq \ell$. The following lemma proves that a packing which fulfills properties (1) to (4) provides a solution that is close to the optimum.

Lemma 3.22. *Let $M = m + h$ be the number of used bins and $\epsilon \leq 1/4$. If properties (1) to (4) hold, then at most $\max\{\Lambda, (1 + \mathcal{O}(\epsilon)) \operatorname{OPT}(I, s) + \mathcal{O}(1)\}$ bins are used in the packing.*

Proof. Case 1: There is no bin containing only small items, i. e., $L = m$. Hence all items are packed into $M = L + h = \Lambda$ bins.

Case 2: There are bins containing only small items, i. e., $L < m$. Property (3) implies that the number of queues d is bounded by $d \leq \epsilon m + 4$. Hence the number of buffer bins is bounded by $\epsilon m + 4$ and the number of heap bins $\Phi(B)$ (property (4)) is bounded by $\Phi(B) = \sum_{i=1}^{\ell-1} r_i + \lceil \epsilon\Lambda \rceil - \ell \leq \ell - 1 + \epsilon\Lambda + 1 - \ell = \epsilon\Lambda$ as $r_i \leq 1$. Since $\Lambda < M$, we can bound $\Phi(B)$ by $\Phi(B) < \epsilon M$. The number of normal bins is thus at least $M - (\epsilon m + 5) - (\epsilon M - 1) \geq M - 2\epsilon M - 4 = (1 - 2\epsilon)M - 4$. By property (2) every normal bin has less than $\epsilon/14$ free space and the total size S of all items is thus at least

$S \geq (1-\epsilon/14)(1-2\epsilon)M-4$. Since $\text{OPT}(I,s) \geq S$, we have $\text{OPT}(I,s) \geq (1-\epsilon/14(1-2\epsilon)M-4$. A simple calculation shows that $\frac{1}{(1-\epsilon/14)(1-2\epsilon)} \leq (1+5\epsilon)$ for $\epsilon \leq 1/4$. Therefore we can bound the number of used bins by $(1+5\epsilon)\,\text{OPT}(I,s)+4$. □

According to property (4) we have to guarantee, that if the rounded potential $\lfloor\Phi(B)\rfloor$ changes, the number of heap bins has to be adjusted accordingly. The potential $\lfloor\Phi(B)\rfloor$ might increases by 1 due to an insert operation. Therefore the number of heap bins has to be incremented. If the potential $\lfloor\Phi(B)\rfloor$ decreases due to a delete operation, the number of heap bins has to be decremented. In order to maintain property (4) we have to make sure, that the number of heap bins can be adjusted whenever $\lfloor\Phi(B)\rfloor$ changes. Therefore we define the fractional part $\{\Phi(B)\} = \Phi(B) - \lfloor\Phi(B)\rfloor$ of $\Phi(B)$ and put it in relation to the fill ratio r_ℓ of bb_ℓ (the last bin containing large items) through the following equation:

$$|(1-r_\ell) - \{\Phi(B)\}| \leq \frac{s}{c(bb_\ell)} \qquad \text{(Heap Equation)}$$

where s is the biggest size of a small item appearing in bb_ℓ. The Heap Equation ensures that the potential $\Phi(B)$ is correlated to $1-r_\ell$. The values may only differ by the small term $\frac{s}{c(bb_\ell)}$. Note that the Heap Equation can always be fulfilled by shifting items from bb_ℓ to queue $Q_{\ell+1}$ or vice versa.

Assuming the Heap Equation holds and the potential $\lfloor\Phi(B)\rfloor$ increases by 1, we can guarantee that buffer bin bb_ℓ is nearly empty. Hence the remaining items can be shifted to $Q_{\ell+1}$ and bb_ℓ can be moved to the heap bins. The bin left of bb_ℓ becomes the new buffer bin of Q_ℓ. Vice versa, if $\lfloor\Phi(B)\rfloor$ decreases, we know by the Heap Equation that bb_ℓ is nearly full, hence we can label bb_ℓ as a normal bin and open a new buffer bin from the heap at the end of queue Q_ℓ. Our goal is to ensure that the Heap Equation is fulfilled at every step of the algorithm along with properties (1) to (4). Therefore we enhance the delete and insert operations from the previous section. Whenever a small item i is inserted or removed, we will perform the operations described in Algorithm 3.19 (which can be applied to bins of different capacities) in the previous section. This will maintain properties (1) to (3). If items are inserted or deleted from queue Q_ℓ (the last queue containing large and small items) the recursion does not halt at bb_ℓ. Instead the recursion goes further and halts at $bb_{\ell+1}$. So, when items are inserted into bin bb_ℓ according to Algorithm 3.19 the bin bb_ℓ is treated as a normal bin. Items are shifted from bb_ℓ to queue $Q_{\ell+1}$ until the Heap Equation is fulfilled. This way we can make sure that the Heap Equation maintains fulfilled whenever an item is inserted or removed from Q_ℓ.

Algorithm 3.23 (Insert or Delete small items for the mixed setting)**.**

Insert(i, b_x, Q_j)**:**

- *Use Algorithm 3.19 to insert item i into Q_j with $j < \ell$.*
- *Let $i_1, \ldots, i_m$ be the items that are inserted at the last step of Algorithm 3.19 into bb_j.*
- *For $k = 1, \ldots, m$ do*
  1. *Insert item i_k into bin bb_j.*
  2. *If bb_j is completely filled use Algorithm 3.20.*
  3. *If the potential $\lfloor\Phi(B)\rfloor$ increases use Algorithm 3.24 (see below) to adjust the number of heap bins (property (4)).*
  4. *Decrease the fill ratio r_ℓ of bb_ℓ by shifting the smallest items in bb_ℓ to $Q_{\ell+1}$ until $(1-r_\ell) \leq \{\Phi(B)\}$ to fulfill the Heap Equation.*

Delete(i, b_x, Q_j)**:**

- *Use Algorithm 3.19 to remove item i from bin b_x in queue Q_j with $j < \ell$.*
- *Let $i_1, \ldots, i_m$ be the items that are removed at the last step of Algorithm 3.19 from bb_j.*
- *For $k = 1, \ldots, m$ do*
  1. *If bb_j is empty use Algorithm 3.20.*
  2. *Remove item i_k from bin bb_j.*
  3. *If the potential $\lfloor \Phi(B) \rfloor$ decreases use Algorithm 3.24.*
  4. *Increase the fill ratio r_ℓ of bb_ℓ by shifting the smallest items in bb_ℓ to $Q_{\ell+1}$ until $(1 - r_\ell) \geq \{\Phi(B)\}$ to fulfill the Heap Equation.*

For the correctness of step 4 (the adjustment to r_ℓ) note the following: In case of the insert operation, the potential $\Phi(B)$ increases and we have $\Phi(B) \geq 1 - r_\ell$. As items are being shifted from bb_ℓ to $Q_{\ell+1}$, the first time that $(1 - r_\ell) \leq \{\Phi(B)\}$ is fulfilled, the Heap Equation is also fulfilled. Since the fill ratio of bb_ℓ changes at most by $\frac{s}{c(bb_\ell)}$ as an item (which has size at most s) is shifted to $Q_{\ell+1}$ we know that $|(1 - r_\ell) - \{\Phi(B)\}| \leq \frac{s}{c(bb_\ell)}$. Correctness of step 4 in the delete operation follows symmetrically.

The potential $\Phi(B)$ changes if items are inserted or deleted into queues $Q_1, \ldots, Q_{\ell-1}$. Due to these insert or delete operations it might happen that the potential $\lfloor \Phi(B) \rfloor$ increases or that a buffer bin is being filled or emptied. The following operation is applied as soon as an item is inserted or deleted into a buffer bin and the potential $\lfloor \Phi(B) \rfloor$ increases or decreases.

Algorithm 3.24 (Change in the potential)**.**

- **Case 1: The potential $\lfloor \Phi(B) \rfloor$ increases by 1.**
  - *According to the Heap Equation the remaining size of small items in bb_ℓ can be bounded. Shift all small items from bb_ℓ to $Q_{\ell+1}$.*
  - *If $|Q_\ell| > 1$ then label the now empty buffer bin bb_ℓ as a heap bin and the last bin in Q_ℓ is labeled as a buffer bin.*
  - *If Q_ℓ only consists of the buffer bin (i. e., $|Q_\ell| = 1$) shift items from $bb_{\ell-1}$ to $Q_{\ell+1}$ until the heap equation is fulfilled. If $bb_{\ell-1}$ becomes empty remove $bb_{\ell-1}$ and bb_ℓ. The bin left to $bb_{\ell-1}$ becomes the new buffer bin of $Q_{\ell-1}$. The queue Q_ℓ is deleted and $Q_{\ell-1}$ becomes the new last queue containing large items.*
- **Case 2: The potential $\lfloor \Phi(B) \rfloor$ decreases by 1.**
  - *According to the Heap Equation the remaining free space in bb_ℓ can be bounded. Shift items from $bb_{\ell+1}$ to bb_ℓ such that the buffer bin bb_ℓ is filled completely.*
  - *Add the new buffer bin from the heap to Q_ℓ.*
  - *If $|Q_\ell| = 1/\epsilon$ label an additional heap bin as a buffer bin to create a new queue $Q_{\ell+1}$ with $|Q_{\ell+1}| = 1$.*

Like in the last section we also have to describe how to handle buffer bins that are being emptied or filled completely. We apply the same algorithm when a buffer bin is being emptied or filled but have to distinguish now between buffer bins of $Q_1, \ldots, Q_\ell$ and buffer bins of $Q_{\ell+1}, \ldots, Q_d$. Since the buffer bins in $Q_{\ell+1}, \ldots, Q_d$ all have capacity 1, we will use the same technique as in the last section. If a buffer bin in $Q_1, \ldots, Q_\ell$ is emptied or filled we will also use similar technique. But instead of inserting a new empty bin as a new buffer bin, we take an existing bin out of the heap. And if a buffer bin from $Q_1, \ldots Q_\ell$ is being emptied (it still contains large items), it is put into the heap. This way we make sure that there are always sufficiently many bins containing large items which are filled completely.

Lemma 3.25. *Let B be an packing which fulfills the properties* (1) *to* (4) *and the Heap Equation. Applying Algorithm 3.24 or Algorithm 3.20 on B during an insert/delete operation yields an packing B' which also fulfills properties* (1) *to* (4)*. The migration to fulfill the Heap Equation is bounded by $\mathcal{O}(1/\epsilon)$.*

Proof. **Analysis of Algorithm 3.24**
Properties (1) and (2) are never violated by the algorithm because the items are only moved by shift operations. Property (3) is never violated because no queue (except for Q_ℓ) exceeds $2/\epsilon$ or falls below $1/\epsilon$ by construction. Algorithm 3.24 is called during an insert or delete operation. The Algorithm is executed as items are shifted into or out of buffer bb_j such that $\lfloor \Phi(B) \rfloor$ changes.

In the following we prove property (4) for the packing B' assuming that $\lfloor \Phi(B) \rfloor = h(B)$ holds by induction. Furthermore we give a bound for the migration to fulfill the heap equation:

- Case 1: The potential $\lfloor \Phi(B) \rfloor$ increases during an insert operation, i. e., it holds $\lfloor \Phi(B') \rfloor = \lfloor \Phi(B) \rfloor + 1$. Let item i^* be the first item that is shifted into a bin bb_j such that $\lfloor \Phi(B) + r^* \rfloor = \lfloor \Phi(B') \rfloor$, where r^* is the fill ratio being added to bb_j by item i^*. In this situation, the fractional part changes from $\{\Phi(B)\} \approx 1$ to $\{\Phi(B')\} \approx 0$.

  - In the case that $|Q_\ell| > 1$, the buffer bin bb_ℓ is being emptied and moved to the heap bins. The bin left of bb_ℓ becomes the new buffer bin bb'_ℓ of Q_ℓ. Hence the number of heap bins increases and we have $h(B') = h(B) + 1 = \lfloor \Phi(B) \rfloor + 1 = \lfloor \Phi(B') \rfloor$, which implies property (4).

    To give a bound on the total size of items needed to be shifted out of (or into) bin bb_ℓ to fulfill the heap equation, we bound the term $|(1 - r'_\ell) - \{\Phi(B')\}|$ by some term $C \leq \mathcal{O}(s(i)/\epsilon)$, where r'_ℓ is the fill ratio of bb'_ℓ and $s(i)$ is the size of the arriving or departing item. If the term $|(1 - r'_\ell) - \{\Phi(B')\}|$ can be bounded by C, the fill ratio of bb'_ℓ has to be adjusted to fulfill the heap equation according to the insert and delete operation. This can be done be shifting a total size of at most C items out of (or into) bb'_ℓ.

    The bin bb'_ℓ is completely filled by property (3) and therefore has a fill ratio of $r'_\ell \geq \frac{c(bb_\ell) - s}{c(bb_\ell)} \geq 1 - 2\frac{s}{\epsilon}$, where $s \leq \frac{\epsilon}{2^k}$ is the largest size of a small item appearing in bb_ℓ and S_k is the largest size category appearing in bb'_ℓ. Let k' be the largest size category appearing in bin bb_j. As the bin bb'_ℓ is right of bb_j we know $k \leq k'$ (property (1)) and hence $s \leq 2s(i^*)$. We get $r'_\ell \geq 1 - 4\frac{s(i^*)}{\epsilon}$. Using that $\{\Phi(B')\} \leq r^* \leq 2^{s(i^*)}/\epsilon$, we can bound $|(1 - r'_\ell) - \{\Phi(B')\}|$ by $4\frac{s(i^*)}{\epsilon} + 2^{s(i^*)}/\epsilon = \mathcal{O}(s(i^*)/\epsilon)$. Hence the Heap Equation can be fulfilled by shifting items of total size $\mathcal{O}(s(i^*)/\epsilon)$ at the end of the insert operation.

  - If $|Q_\ell| = 1$ a set of items in the buffer bin $bb_{\ell-1}$ is shifted to $Q_{\ell+1}$ to fulfill the Heap Equation. Since items are being removed from $bb_{\ell-1}$ the potential decreases. If $r_{\ell-1} > \{\Phi(B')\}$, there are enough items which can be shifted out of $bb_{\ell-1}$ such that we obtain a new potential $\Phi(B'') < \Phi(B') - \{\Phi(B')\}$. Hence $\lfloor \Phi(B'') \rfloor = \lfloor \Phi(B) \rfloor$ and the Heap Equation is fulfilled.

    Note that the size of items that are shifted out of $bb_{\ell-1}$ is bounded by $r^* + s = \mathcal{O}(s(i^*)/\epsilon)$, where s is the biggest size of an item appearing in $bb_{\ell-1}$.

    If $r_{\ell-1} \leq \{\Phi(B')\}$ all items are shifted out of $bb_{\ell-1}$. As the number of queues decreases, we obtain the new potential $\Phi(B'') = \Phi(B') - r_{\ell-1} + 1 = \lfloor \Phi(B') \rfloor +$

$\{\Phi(B')\} - r_{\ell-1} + 1 \geq \lfloor \Phi(B') \rfloor + 1$. Hence $\lfloor \Phi(B'') \rfloor = \lfloor \Phi(B) \rfloor + 2$. The buffer bins $bb_{\ell-1}$ and bb_ℓ are moved to the heap and thus $h(B'') = h(B) + 2 = \lfloor \Phi(B) \rfloor + 2 = \lfloor \Phi(B'') \rfloor$ (property (4)).

Note that if $r_{\ell-1} \leq \{\Phi(B')\}$, item i^* is not inserted into bin $bb_{\ell-1}$ as $r_{\ell-1} \geq r^* > \{\Phi(B')\}$. Therefore the bin bb_j is left of $bb_{\ell-1}$ and we can bound the fill ratio of the bin left of $bb_{\ell-1}$ called r''_ℓ by $1 - 2\frac{s(i^*)}{\epsilon}$. Using $\{\Phi(B'')\} \leq r^* = \mathcal{O}(s(i^*)/\epsilon)$ the heap equation can be fulfilled by shifting items of total size $\mathcal{O}(s(i)/\epsilon)$ at the end of the insert operation.

- Case 2: The potential $\lfloor \Phi(B) \rfloor$ decreases during a delete operation, i. e., it holds $\lfloor \Phi(B') \rfloor = \lfloor \Phi(B) \rfloor - 1 = \lfloor \Phi(B) - r^* \rfloor$, where r^* is the fill ratio being removed from a buffer bin bb_j due to the first shift of an item i^* that decreases the potential. According to Algorithm 3.24, buffer bin bb_ℓ is being filled completely and a new buffer bin for Q_ℓ is inserted from the heap. Hence the number of heap bins decreases and we have $\lfloor \Phi(B') \rfloor = h(B) - 1 = h(B')$.

  As $\lfloor \Phi(B) \rfloor - 1 = \Phi(B) - \{\Phi(B)\} - 1 = \lfloor \Phi(B) - r^* \rfloor$, it holds that $\{\Phi(B)\} \leq r^*$ and by the heap equation the fill ratio of bb_ℓ is $r_\ell \geq r^* + s$, where s is the largest size of a small item in bb_ℓ. As above, r^* and s can be bounded by $\mathcal{O}(\frac{s(i^*)}{\epsilon})$. Hence the total size that is shifted from $Q_{\ell+1}$ into bin bb_ℓ can be bounded by $\mathcal{O}(\frac{s(i^*)}{\epsilon})$.

  Furthermore $\{\Phi(B')\} \geq 1 - r^*$ (as $\Phi(B') = \Phi(B) - r^*$) and $r'_\ell = 0$, therefore we can bound $|(1 - r'_\ell) - \{\Phi(B')\}|$ by $r^* \leq \mathcal{O}(s(i^*)/\epsilon)$ and the Heap Equation can be fulfilled by shifting a total size of at most $\mathcal{O}(s(i^*)/\epsilon)$ items.

  In the case that $|Q_\ell| = 1/\epsilon$ a new queue $Q_{\ell+1}$ is created which consists of a single buffer bin (inserted from the heap), which does not contain small items, i. e., $h(B'') = h(B') - 1 = h(B) - 2$, where B'' is the packing after the insertion of item i^*. Let $\Phi(B'')$ be the potential after the queue $Q_{\ell+1}$ is created. Then $\Phi(B'') = \sum_{i=1}^{\ell(B'')-1} r_i + \epsilon\Lambda - \ell(B'') = \sum_{i=1}^{\ell(B')-2} r_i + \epsilon\Lambda - \ell(B') - 1 = \Phi(B') - 1$, as the buffer bin bb_ℓ is now counted in the potential, but does not contain any small items and thus $r''_\ell = 0$. Hence $\Phi(B'') = \Phi(B') - 1 = h(B') - 1 = h(B'')$.

Analysis of Algorithm 3.20

Algorithm 3.20 is executed as an item i^* is moved into a buffer bin bb_j such that bb_j is completely filled or Algorithm 3.20 is executed if the buffer bin bb_j is emptied by moving the last item i^* out of the bin. As in the analysis of Algorithm 3.24, properties (1) and (2) are never violated by the algorithm because the items are only moved by shift operations. Property (3) is never violated because no queue (except for Q_ℓ) exceeds $2/\epsilon$ or falls below $1/\epsilon$ by construction.

It remains to prove property (4) and a bound for the migration to fulfill the heap equation:

- Case 1: An item i^* is moved into the buffer bin bb_j such that bb_j is filled completely for some $j < \ell$. According to Algorithm 3.20 a bin is taken out of the heap and labeled as the new buffer bin bb'_j with fill ratio $r'_j = 0$ of queue Q_j, i. e., the number of heap bins decreases by 1. Let $\Phi(B)$ be the potential before Algorithm 3.20 is executed and let $\Phi(B')$ be the potential after Algorithm 3.20 is executed. The potential changes as follows:

$$\Phi(B) - \Phi(B') = (r_j - r'_j) - (\ell(B) - \ell(B'))$$

Since $r'_j = 0$ the new potential is $\Phi(B') = \Phi(B) - r_j \approx \Phi(B) - 1$ (assuming $\ell(B) = \ell(B')$, as the splitting of queue is handled later on).

- If $\lfloor \Phi(B') \rfloor = \lfloor \Phi(B) \rfloor - 1$ property (4) is fulfilled since the number of heap bins decreases by $h(B') = h(B) - 1 = \lfloor \Phi(B) \rfloor - 1 = \lfloor \Phi(B') \rfloor$. As $r_j \geq \frac{c(bb_j)-s}{c(bb_j)}$, where s is the biggest size category appearing in bb_j and $s \leq 2s(i^*)$, we obtain for the fractional part of the potential that $\{\Phi(B)\} - \{\Phi(B')\} \leq 2\frac{s}{\epsilon} \leq 4\frac{s(i^*)}{\epsilon}$. Hence the Heap Equation can be fulfilled by shifting items of total size $\mathcal{O}(s(i^*)/\epsilon)$ at the end of the insert operation as in the above proof.
- In the case that $\lfloor \Phi(B') \rfloor = \lfloor \Phi(B) \rfloor = \lfloor \Phi(B) - r_j \rfloor$ we know that the fractional part changes by $\{\Phi(B')\} = \{\Phi(B)\} - r_j$. Since the bin bb_j is filled completely we know that $r_j \geq \frac{c(bb_j)-s}{c(bb_j)} \approx 1$ and hence $\{\Phi(B)\} \geq r_j \approx 1$ and $\{\Phi(B')\} \leq 1 - r_j \approx 0$. According to the Heap Equation, items have to be shifted out of r_ℓ such that the fill ratio r_ℓ changes from $r_\ell \leq 1 - r_j$ to $r_\ell \approx 1$. Therefore we know that as items are shifted out of bb_ℓ to fulfill the Heap Equation, the buffer bin bb_ℓ is being emptied and moved to the heap (see Algorithm 3.24). We obtain for the number of heap bins that $h(B') = h(B) + 1 - 1 = h(B)$ and hence $h(B') = \lfloor \Phi(B') \rfloor$ (property (4)).

  As $\{\Phi(B)\} \geq r_j \geq 1 - 4\frac{s(i^*)}{\epsilon}$, the Heap Equation implies that $r_\ell \leq 4\frac{s(i^*)}{\epsilon} + \frac{s}{c(bb_\ell)} = \mathcal{O}(s(i^*)/\epsilon)$. The buffer bin bb_ℓ is thus emptied by moving a size of $\mathcal{O}(s(i^*)/\epsilon)$ items out of the bin. Let bb'_ℓ be the new buffer bin of Q_ℓ that was left of bb_ℓ. The Heap Equation can be fulfilled by shifting at most $\mathcal{O}(s(i)/\epsilon)$ out of bb'_ℓ since $\{\Phi(B')\}$ is bounded by $1 - r_j = \mathcal{O}(s(i^*)/\epsilon)$.
- In the case that $|Q_j| > 2/\epsilon$ the queue is split into two queues and an additional heap bin is inserted, i. e., $h(B'') = h(B') - 1$. As the potential changes by $\Phi(B'') = \Phi(B') + (\ell(B') - \ell(B'')) = \Phi(B') - 1$ we obtain again that $h(B'') = \lfloor \Phi(B'') \rfloor$.

- Case 2: Algorithm 3.20 is executed if bin bb_j is emptied due to the removal of an item i^* as a result of a Delete(i, b_x, Q_j) call. According to Algorithm 3.20, the emptied bin is moved to the heap, i. e., the number of heap bins increases by 1. Depending on the length of Q_j and Q_{j+1}, the bin right of bb_j or the bin left of bb_j is chosen as the new buffer bin bb'_j. The potential changes by $\Phi(B') = \Phi(B) + r'_j$, where r'_j is the fill ratio of bb'_j as in case 1.
  - If $\lfloor \Phi(B') \rfloor = \lfloor \Phi(B) \rfloor + 1$ property (4) is fulfilled since the number of heap bins increases by $h(B') = h(B) + 1$.

    As bin bb'_j is completely filled, the fill ratio is bounded by $r'_j \geq 1 - 2\frac{s}{\epsilon}$, where s is the largest size appearing in bb'_j. Since the bin b_x has to be left of bb_j we know that $s \leq 2s(i)$. We obtain for the fractional part of the potential that $\{\Phi(B)\} \geq \{\Phi(B')\} - 2\frac{s}{\epsilon} \leq 4\frac{s(i)}{\epsilon}$. Hence the Heap Equation can be fulfilled by shifting items of total size $\mathcal{O}(s(i)/\epsilon)$ at the end of the remove operation.
  - In the case that $\lfloor \Phi(B') \rfloor = \lfloor \Phi(B) \rfloor = \lfloor \Phi(B) + r'_j \rfloor$ we know that the fractional part changes similar to case 1 by $\{\Phi(B')\} = \{\Phi(B)\} + r'_j$. Since the bin bb_j is filled completely we know that $r_j \geq \frac{c(bb_j)-s}{c(bb_j)} \approx 1$ and hence $\{\Phi(B')\} \geq r_j \approx 1$ and $\{\Phi(B)\} \leq 1 - r_j \approx 0$. According to the Heap Equation items have to be shifted to bb_ℓ such that the fill ratio r_ℓ changes from $r_\ell \approx 0$ to $r_\ell \approx 1$. Therefore

we know that as items are shifted into bb_ℓ to fulfill the Heap Equation, bb_ℓ is filled completely and a bin from the heap is labeled as the new buffer bin of Q_ℓ (see Algorithm 3.24). We obtain for the number of heap bins that $h(B') = h(B) - 1 + 1 = h(B)$ and hence $h(B') = \Phi(B')$ (property (4)). The Heap Equation can be fulfilled similarly to case 1 by shifting items of total size $\mathcal{O}(s(i)/\epsilon)$.

□

Using the above lemma for, we can finally prove the following central theorem, which states that the migration of an insert/delete operation is bounded and that properties (1) to (4) are maintained.

Theorem 3.26.

(i) Let B be a packing which fulfills properties (1) *to* (4) *and the Heap Equation. Applying operations insert(i, b_x, Q_j) or delete(i, b_x, Q_j) on a packing B yields an instance B' which also fulfills properties* (1) *to* (4) *and the Heap Equation.*

(ii) The migration factor of an insert/delete operation is bounded by $\mathcal{O}(1/\epsilon)$.

Proof. Suppose a small item i with size $s(i)$ is inserted or deleted from queue Q_j. The insert and delete operation basically consists of application of Algorithm 3.19 and iterated use of steps (1) to (3) where Algorithms 3.20 and 3.24 are used and items in bb_ℓ are moved to $Q_{\ell+1}$ and vice versa. Let B be the packing before the insert/delete operation and let B' be the packing after the operation.

Proof for (i): Now suppose by induction that property (1) to (4) and the Heap Equation is fulfilled for packing B. We prove that property (4) and the Heap Equation maintain fulfilled after applying an insert or delete operation on B resulting in the new packing B'. Properties (1) to (3) hold by conclusion of Lemma 3.21 and Lemma 3.25. Since the potential and the number of heap bins only change as a result of Algorithm 3.20 or Algorithm 3.24, property (4) maintains fulfilled also. By definition of step 4 in the insert operation, items are shifted from bb_ℓ to $Q_{\ell+1}$ until the Heap Equation is fulfilled. By definition of step 4 of the delete operation, the size of small items in bb_ℓ is adjusted such that the Heap Equation is fulfilled. Hence the Heap Equation is always fulfilled after application of Insert(i, b_x, Q_j) or Delete(i, b_x, Q_j).

Proof for (ii): According to Lemma 3.21 the migration factor of the usual insert operation is bounded by $\mathcal{O}(1/\epsilon)$. By Lemma 3.25 the migration in Algorithm 3.20 and Algorithm 3.24 is also bounded by $\mathcal{O}(1/\epsilon)$. It remains to bound the migration for step 4 in the insert/delete operation. Therefore we have to analyze the total size of items to be shifted out or into bb_ℓ in order to fulfill the Heap Equation.

Since the size of all items $i_1, \ldots, i_k$ that are inserted into bb_j is bounded by $7s(i)$ (see Lemma 3.21) and the capacity of bb_j is at least $\epsilon/14$ the potential $\Phi(B)$ changes by at most $\mathcal{O}(s(i)/\epsilon)$. By Lemma 3.25 the size of items that needs to be shifted out or into bb_ℓ as a result of Algorithm 3.20 or 3.24 is also bounded by $\mathcal{O}(s(i)/\epsilon)$. Therefore the size of all items that need to be shifted out or into bb_ℓ in step (4) of the insert/delete operation is bounded by $\mathcal{O}(s(i)/\epsilon)$.

Shifting a size of $\mathcal{O}(s(i)/\epsilon)$ to $Q_{\ell+1}$ or vice versa leads to a migration factor of $\mathcal{O}(1/\epsilon^2)$ (Lemma 3.21). Fortunately we can modify the structure of queues $Q_{\ell+1}$ and $Q_{\ell+2}$ such that we obtain a smaller migration factor. Assuming that $Q_{\ell+1}$ consists of a single buffer

bin, i. e., $|Q_{\ell+1}| = 1$ items can directly be shifted from bb_ℓ to $bb_{\ell+1}$ and therefore we obtain a migration factor of $\mathcal{O}(1/\epsilon)$. A structure with $|Q_{\ell+1}| = 1$ and $1 \leq |Q_{\ell+2}| \leq 2/\epsilon$ (see property (3)) can be maintained by changing Algorithm 3.20 in the following way:

- If $bb_{\ell+1}$ is filled completely, move the filled bin to $Q_{\ell+2}$.
  - If $|Q_{\ell+2}| > 2/\epsilon$, split $Q_{\ell+2}$ into two queues.
- If $bb_{\ell+1}$ is being emptied, remove the bin and label the first bin of $Q_{\ell+2}$ as $bb_{\ell+1}$.
  - If $|Q_{\ell+2}| = 0$, remove $Q_{\ell+2}$.

□

3.4.3 Handling the General Setting

In the previous section we described how to handle small items in a mixed setting. It remains to describe how large items are handled in this mixed setting. Algorithm 3.7 describes how to handle large items only. However, in a mixed setting, where there are also small items, we have to make sure that properties (1) to (4) and the Heap Equation maintain fulfilled as a large item is inserted or deleted. Algorithm 3.7 changes the configuration of at most $\mathcal{O}(1/\epsilon^2 \cdot \log 1/\epsilon)$ bins (Theorem 3.16). Therefore, the size of large items in a bin b $(= 1 - c(b))$ changes, as Algorithm 3.7 may increase or decrease the capacity of a bin. Changing the capacity of a bin may violate properties (2) to (4) and the Heap Equation. We describe an algorithm to change the packing of small items such that all properties and the Heap Equation are fulfilled again after Algorithm 3.7 was applied.

The following algorithm describes how the length of a queue Q_j is adjusted if the length $|Q_j|$ falls below $1/\epsilon$:

Algorithm 3.27 (Adjust the queue length)**.**

- *Remove all small item I_S from bb_j and add bb_j to the heap.*
- *Merge Q_j with Q_{j+1}. The merged queue is called Q_j.*
- *If $|Q_j| > 2/\epsilon$ split queue Q_j by adding a heap bin in the middle.*
- *Insert items I_S using Algorithm 3.23.*

The following algorithm describes how the number of heap bins can be adjusted.

Algorithm 3.28 (Adjust number of heap bins)**.**

- *Decreasing the number of heap bins by* 1*.*
  - *Shift small items from $Q_{\ell+1}$ to bb_ℓ until bb_ℓ is filled completely*
  - *Label a heap bin as the new buffer bin of Q_ℓ*
- *Increasing the number of heap bins by* 1*.*
  - *Shift all small items from bb_ℓ to $Q_{\ell+1}$*
  - *Label bb_ℓ as a heap bin*
  - *Label the bin left of bb_ℓ as new buffer bin of Q_ℓ*

Note that the Heap Equation can be fulfilled in the same way, by shifting items from bb_ℓ to $Q_{\ell+1}$ or vice versa.

Using these algorithms, we obtain our final algorithm for the fully dynamic *binpacking* problem.

Algorithm 3.29 (AFPTAS for the mixed setting)**.**

- *If i is large do*
  1. *Use Algorithm 3.7.*
  2. *Remove all small items I_S of bins b with changed capacity.*
  3. *Adjust queue length.*
  4. *Adjust the number of heap bins.*
  5. *Adjust the Heap Equation.*
  6. *Insert all items I_S using Algorithm 3.23.*
- *If i is small use Algorithm 3.23*

Combining all the results from the current and the previous section, we finally prove the central result that there is fully dynamic AFPTAS for the *binpacking* problem with polynomial migration.

Theorem 3.30. *Algorithm 3.29 is a fully dynamic AFPTAS for the binpacking problem, that achieves a migration factor of at most $\mathcal{O}(1/\epsilon^4 \cdot \log 1/\epsilon)$ by repacking items from at most $\mathcal{O}(1/\epsilon^3 \cdot \log 1/\epsilon)$ bins.*

Proof. **Approximation guarantee:** By definition of the algorithm, it generates at every timestep t a packing B_t of instance $I(t)$ such that properties (1) to (4) are fulfilled. According to Lemma 3.22, at most $\max\{\Lambda, (1+\mathcal{O}(\epsilon))\,\text{OPT}(I(t), s) + \mathcal{O}(1)\}$ bins are used where Λ is the number of bins containing large items. Since we use Algorithm 3.7 to pack the large items, Theorem 3.16 implies that $\Lambda \leq (1+\mathcal{O}(\epsilon))\,\text{OPT}(I(t), s) + \mathcal{O}(1/\epsilon \log 1/\epsilon)$. Hence the number of used bins can be bounded in any case by $(1+\mathcal{O}(\epsilon))\,\text{OPT}(I(t), s) + \mathcal{O}(1/\epsilon \log 1/\epsilon)$.

Migration Factor: Note that the Algorithm uses Algorithm 3.23 or Algorithm 3.7 to insert and delete small or large items. The migration factor for Algorithm 3.23 is bounded by $\mathcal{O}(1/\epsilon)$ due to Theorem 3.26 while the migration factor for Algorithm 3.7 is bounded by $\mathcal{O}(1/\epsilon^3 \cdot \log 1/\epsilon)$ due to Theorem 3.16.

It remains to bound the migration that is needed to adjust the heap bins, the length of a queue falling below $1/\epsilon$ and the Heap Equation in case a large item arrives and Algorithm 3.7 is applied.

Suppose the number of heap bins has to be adjusted by 1. In this case Algorithm 3.28 shifts items from $Q_{\ell+1}$ to bb_ℓ or vice versa until bb_ℓ is either filled or emptied. Hence, the size of moved items is bounded by 1. Since the size of the arriving or departing item is $\geq \epsilon/14$ the migration factor is bounded by $\mathcal{O}(1/\epsilon)$. In the same way, a migration of at most $\mathcal{O}(1/\epsilon)$ is used to fulfill the Heap Equation which implies that the migration in step 5 is bounded by $\mathcal{O}(1/\epsilon)$.

If $|Q_j|$ falls below $1/\epsilon$, the two queues Q_j and Q_{j+1} are merged by emptying bb_j. The removed items are inserted by Algorithm 3.23. As their total size is bounded by 1 and the algorithm has a migration factor of $\mathcal{O}(1/\epsilon)$, the size of the moved items is bounded by $\mathcal{O}(1/\epsilon)$. The migration to merge two queues can thus be bounded by $\mathcal{O}(1/\epsilon^2)$.

Note that the proof of Theorem 3.16 implies that at most $\gamma = \mathcal{O}(1/\epsilon^2 \log 1/\epsilon)$ bins are changed by Algorithm 3.7. The total size of the items I_S which are removed in step 2 is thus bounded by γ. Similarly, the length of at most γ queues can fall below $1/\epsilon$. The migration of step 3 is thus bounded by $\gamma \cdot 1/\epsilon^2$. As at most γ buffer bins are changed, the change of the potential (and thus the number of heap bins) is also bounded by γ and the migration in step 4 can be bounded by $\gamma \cdot 1/\epsilon$. The migration in step 6 is bounded by $s(I_S) \cdot 1/\epsilon \leq \gamma \cdot 1/\epsilon$ as Algorithm 3.23 has migration factor $1/\epsilon$. The total migration of the adjustments is thus bounded by $\gamma \cdot 1/\epsilon^2 = \mathcal{O}(1/\epsilon^4 \log 1/\epsilon)$.

Running Time: The handling of small items can be performed in linear time while the handling of large items requires $\mathcal{O}(M(1/\epsilon \log(1/\epsilon)) \cdot 1/\epsilon^3 \log(1/\epsilon) + 1/\epsilon \log(1/\epsilon) \log(\epsilon^2 \cdot n(t)) + \epsilon n(t))$, where $M(n)$ is the time needed to solve a system of n linear equations (see Theorem 3.16). The total running time of the algorithm is thus $\mathcal{O}(M(1/\epsilon \log(1/\epsilon)) \cdot 1/\epsilon^3 \log(1/\epsilon) + 1/\epsilon \log(1/\epsilon) \log(\epsilon^2 \cdot n(t)) + n(t))$. □

4 Closing the Gap for Makespan Scheduling via Sparsification Techniques

4.1 Introduction

Minimum makespan scheduling is one of the foundational problems in the literature on approximation algorithms [Gra66; Gra69]. In the *identical machine* setting the problem asks for an assignment of a set of n jobs $\mathcal{J}$ to a set of m identical machines $\mathcal{M}$. Each job $j \in \mathcal{J}$ is characterized by a non-negative processing time $p_j \in \mathbb{Z}_{>0}$. The load of a machine is the total processing time of jobs assigned to it, and our objective is to minimize the *makespan*, that is, the maximum machine load. This problem is usually denoted $P||C_{\max}$. It is well known to admit a *polynomial time approximation scheme* (PTAS) [HS87], and there has been many subsequent works improving the running time or deriving PTAS's for more general settings. The fastest PTAS for $P||C_{\max}$ has a running time of $2^{O(1/\varepsilon^2)\log^3(1/\varepsilon))} + O(n \log n)$ for $(1+\varepsilon)$-approximate solutions [Jan10]. Very recently, Chen et al. [CJZ13] showed that, assuming the *exponential time hypothesis* (ETH), there is no PTAS that yields $(1+\varepsilon)$-approximate solutions for $\varepsilon > 0$ with running time $2^{(1/\varepsilon)^{1-\delta}} + \text{poly}(n)$ for any $\delta > 0$ [CJZ13].

Given a guess $T \in \mathbb{N}$ on the optimal makespan, which can be found with binary search, the problem reduces to deciding the existence of a packing of the jobs to m machines (or bins) of capacity T. If we aim for a $(1+\varepsilon)$-approximate solution, for some $\varepsilon > 0$, we can assume that all processing times are integral and T is a constant number, namely $T \in O(1/\varepsilon^2)$. This can be achieved with well known rounding and scaling techniques [Alo+97; Alo+98; Hoc97] which will be specified later. Let $\pi_1 < \pi_2 < \ldots < \pi_d$ be the job sizes appearing in the instance after rounding, and let b_k denote the number of jobs of size π_k. The mentioned rounding procedure implies that the number of different job sizes is $d = O((1/\varepsilon)\log(1/\varepsilon))$. Hence, for large n we obtain a highly symmetric problem where several jobs will have the same processing time. Consider the *knapsack polytope* $\mathcal{P} = \{c \in \mathbb{R}^d_{\geq 0} : \pi \cdot c \leq T\}$. A packing on one machine can be expressed as a vector $c \in Q = \mathbb{Z}^d \cap \mathcal{P}$, where c_k denotes the number of jobs of size π_k assigned to the machine. Elements in $Q = \mathbb{Z}^d \cap \mathcal{P}$ are called *configurations*. Considering a variable $x_c \in \mathbb{Z}_{\geq 0}$ that decides the multiplicity of configuration c in the solution, our problem reduces to solving the following linear integer program (ILP):

$$\text{[conf-IP]} \quad \sum_{c \in Q} c \cdot x_c = b, \tag{4.1}$$

$$\sum_{c \in Q} x_c = m, \tag{4.2}$$

$$x_c \in \mathbb{Z}_{\geq 0} \qquad \text{for all } c \in Q. \tag{4.3}$$

In this article we derive new insights on this ILP that help us to design faster algorithms for $P||C_{\max}$ and other more general problems. These including makespan scheduling on *related machines* $Q||C_{\max}$, and a more general class of objective functions on parallel machines. We show that all these problems admit a PTAS with running time $2^{O((1/\varepsilon)\log^4(1/\varepsilon))} + \text{poly}(n)$. Hence, our algorithm is best possible up to polylogarithmic factors in the exponent assuming ETH [CJZ13].

4.1.1 Literature Review

There is an old chain of approximation algorithms for $P||C_{\max}$, starting from the seminal work by Graham [Gra66; Gra69]. The first PTAS was given by Hochbaum and Shmoys [HS87] and had a running time of $(n/\varepsilon)^{O((1/\varepsilon)^2)} = n^{O((1/\varepsilon)^2\log(1/\varepsilon))}$. This was improved to $n^{O((1/\varepsilon)\log^2(1/\varepsilon))}$ by Leung [Leu89]. Subsequent articles improve further the running time. In particular Hochbaum and Shmoys (see [Hoc97]) and Alon et al. [Alo+97; Alo+98] obtain an *efficient PTAS*[1] (EPTAS) with running time $2^{(1/\varepsilon)^{\text{poly}(1/\varepsilon)}} + O(n\log n)$. Alon et al. [Alo+97; Alo+98] consider general techniques that work for several objective functions, including all L_p-norm of the loads and maximizing the minimum machine load.

The fastest PTAS known up to date for $P||C_{\max}$ achieves a running time of $2^{O((1/\varepsilon)^2\log^3(1/\varepsilon))} + O(n\log n)$ [Jan10]. More generally, this work gives an EPTAS for the case of related (uniform) machines, where each machine $i \in \mathcal{M}$ has a speed s_i and assigning to i job j implies a processing time of p_j/s_i. For this more general case the running time is $2^{O((1/\varepsilon)^2\log^3(1/\varepsilon))} + \text{poly}(n)$. For the simpler case of $P||C_{\max}$, the ILP can be solved directly since the number of variables is a constant. This can be done with Lentras' algorithm [Len83], or even with Kannan's algorithm [Kan87] that gives an improved running time. This technique yields a running time that is doubly exponential in $1/\varepsilon$. This was, in essence, the approach by Alon et al. [Alo+97; Alo+98] and Hochbaum and Shmoys [Hoc97]. To lower the dependency on $1/\varepsilon$, Jansen [Jan10] uses a result by Eisenbrand and Shmonin [ES06] that implies the existence of a solution x with support of size at most $O(d\log(dT)) = O((1/\varepsilon)\log^2(1/\varepsilon))$. First guessing the support and then solving the ILP with $O((1/\varepsilon)\log^2(1/\varepsilon))$ integer variables and using Kannan's algorithm yields the desired running time of $2^{O((1/\varepsilon)^2\log^3(1/\varepsilon))} + O(n\log n)$.

The configuration ILP has recently been studied in the context of the (1-dimensional) cutting stock problem. In this case, the dimension d is constant, $T = 1$, and π is a rational vector. Moreover, π and b are part of the input. Goemans and Rothvoß [GR14] obtain an optimal solution in time $\log(\Delta)^{2^{O(d)}}$, where Δ is the largest number appearing in the denominator of π_k or the multiplicities b_k. This is achieved by first showing that there exists a pre-computable set $\tilde{Q} \subseteq Q$ with polynomial many elements, such that there exists a solution x that gives all but constant (depending only on d) amount of weight to $\tilde{Q}$. We remark that applying this result to a rounded instance of $P||C_{\max}$ yields a running time that is doubly exponential on $1/\varepsilon$.

4.1.2 Our Contributions

Our main contribution is a new insight on the structure of the solutions of [conf-IP]. These properties are specially tailored to problems in which T is bounded by a constant, which

[1] That is, a PTAS whose running time is $f(1/\varepsilon)\text{poly}(|I|)$ where $|I|$ is the encoding size of the input and f is some function.

in the case of $P||C_{\max}$ can be guaranteed by rounding and scaling. The same holds for $Q||C_{\max}$ with a more complex rounding and case analysis.

We first classify configurations by their support. We say that a configuration is *simple* if its support is of size at most $\log(T+1)$, otherwise it is *complex*. Our main structural result[2] states that there exists a solution x in which all but $O(d\log(dT))$ weight is given to simple configurations, the support is bounded by $O(d\log(dT))$ (as implied by Eisenbrand and Shmonin [ES06]) and no complex configuration has weight larger than 1.

Theorem 4.1 (Thin solutions)**.** *Assume that [conf-IP] is feasible. Then there exists a feasible solution x to [conf-IP] such that:*

1. *if $x_c > 1$ then the configuration c is simple,*
2. *the support of x satisfies $|\operatorname{supp}(x)| \leq 4(d+1)\log(4(d+1)T)$, and*
3. *$\sum_{c\in Q_c} x_c \leq 2(d+1)\log(4(d+1)T)$, where Q_c denotes the set of complex configurations.*

We call a solution satisfying the properties of the theorem *thin*. The theorem can be shown by iteratively applying a sparsification lemma that shows that if a solution gives a weight of two or more to a complex configuration, then we can replace this partial solution by two configurations with smaller support. The sparsification lemma is shown by a simple application of the pigeonhole principle. The theorem can be shown by mixing this technique with the theorem of Eisenbrand and Shmonin [ES06] and a potential function argument.

As an application to our main structural theorem, we derive a PTAS for $P||C_{\max}$ by first guessing the jobs assigned to complex configurations. An optimal solution for this subinstance can be derived by a dynamic program. For the remaining instance we know the existence of a solution using only simple configurations. Then we can guess the support of such solution and solve the corresponding [conf-IP] restricted to the guessed variables. The main use of having simple configurations is that we can guess the support of the solution much faster, as the number of simple configuration is (asymptotically) smaller than the total number of configurations. The complete procedure takes time $2^{O((1/\varepsilon)\log^4(1/\varepsilon))} + O(n\log n)$. Moreover, using the rounding and case analysis of Jansen [Jan10], we derive an mixed integer linear program that can be suitably decomposed in order to apply our structural result iteratively. This yields a PTAS with a running time of $2^{O((1/\varepsilon)\log^4(1/\varepsilon))} + \operatorname{poly}(n)$ for $Q||C_{\max}$.

Similarly, we can extend our results to derive PTAS's for a larger family of objective functions as considered by Alon et al. [Alo+97; Alo+98]. Let ℓ_i denote the load of machine i, that is, the total processing time of jobs assigned to machine i for a given solution. Our techniques then gives a PTAS with the same running time for the problem of minimizing the L_p-norms of the loads (for fixed p), and maximizing $\min_{i\in M}\ell_i$, among others. To solve this problem, we can round the instance and state an IP analogous to [conf-IP] but considering an objective function. However, the objective function prevents us to use the main theorem as it is stated. To get over this issue, we study several ILPs. In each ILP we consider x_c to be a variable only if c has a given load, and fix the rest to be

[2]We remark the resemblance of this structure to the result by Goemans and Rothvoß [GR14]. Indeed, similar to their result, we can precompute a subset of configurations such that all but a constant amount of weight of the solution is given to such set. In their case the set is of cardinality polynomial on the input and is constructed by covering the integral solutions of the knapsack polytope by parallelepipeds. In our case, all but $O(d\log dT)$ weight is given to simple configurations.

some optimal solution. Applying to each such ILP Theorem 4.1, plus some extra ideas, yields an analogous structural theorem. Afterwards, an algorithm similar to the one for makespan minimization yields the desired PTAS.

From an structural point of view, our sparsification lemma has other consequences on the structure of the knapsack polytope and the LP-relaxation of the [conf-IP]. More precisely, we can show that any vertex of the convex hull of Q must be simple. This, for example, helps us to upper bound the number of vertices by $2^{O(\log^2(T)+\log^2(d))}$. Moreover, we can show that the configuration-LP, obtained by replacing the integrality restriction in [conf-IP] by $x \geq 0$, if it is feasible then admits a solution whose support consist purely of simple configurations. Due to space limitations we leave many details and proofs to the appendix.

4.2 Preliminaries

We will use the following notation throughout the chapter. By default $\log(\cdot) = \log_2(\cdot)$, unless stated otherwise. Given two sets A, I, we will denote by A^I the set of all vectors indexed by I with entries in A, that is, $A^I = \{(a_i)_{i\in I} : a_i \in A \text{ for all } i \in I\}$. Moreover, for $A \subseteq \mathbb{R}$, we denote the support of a vector $a \in A^I$ as $\operatorname{supp}(a) = \{i \in I : a_i \neq 0\}$.

We consider an arbitrary knapsack polytope $\mathcal{P} = \{c \in \mathbb{R}^d_{\geq 0} : \pi \cdot c \leq T\}$ where $\pi \in \mathbb{Z}^d_{>0}$ is a non-negative integral (row) vector and T is a positive integer. We assume without loss of generality that each coordinate π_k of π is upper bounded by T (otherwise $c_k = 0$ for all $c \in \mathbb{Z}^d \cap \mathcal{P}$). We focus on the set of integral vectors in $\mathcal{P}$ which we denote by $Q = \mathbb{Z}^d \cap \mathcal{P}$. We call an element $c \in Q$ a *configuration*. Given $b \in \mathbb{R}^d$, consider the problem of decomposing b as a conic integral combination of m configurations. That is, our aim is to find a feasible solution to [conf-IP], defined above.

A crucial property of the [conf-IP] is that there is always a solution with a support of small cardinality. This follows from a Caratheodory-type bound obtained by Eisenbrand and Shmonin [ES06]. Since we will need the argument later, we state the result applied to our case and revise its (very elegant) proof. We split the proof in two lemmas.

For a given subset $A \subseteq Q$, let us denote by x^A the indicator vector of A, that is $x^A_c = 1$ if $c \in A$, and 0 otherwise. Let us also denote by M the $(d+1) \times |Q|$ matrix that defines the system of equalities (4.1) and (4.2).

Lemma 4.2 (Eisenbrand and Shmonin [ES06])**.** *Let $x \in \mathbb{Z}^Q_{\geq 0}$ be a vector such that $|\operatorname{supp}(x)| > 2(d+1)\log(4(d+1)T)$. Then there exist two disjoint sets A, B with $\emptyset \neq A, B \subseteq \operatorname{supp}(x)$ such that $Mx^A = Mx^B$.*

Proof. Let $s := |\operatorname{supp}(x)|$. Each coordinate of M is smaller than T. Hence, for any $A \subseteq \operatorname{supp}(x)$, each coordinate of Mx^A is no larger than $|A| \cdot T \leq sT$. Thus, Mx^A belongs to $\{0, \ldots, sT\}^{d+1}$, and hence there are at most $(sT+1)^{d+1} = 2^{(d+1)\log(sT+1)}$ different possibilities for vector Mx^A, over all possible subsets $A \subseteq \operatorname{supp}(x)$. On the other hand, there are 2^s many different subsets of $\operatorname{supp}(x)$.

We claim that $s > (d+1)\log(sT+1)$. Indeed, since $s > 2(d+1)\log(4(d+1)T)$ then $T < 2^{\frac{s}{2(d+1)}}/(4(d+1))$. Hence,

$$
\begin{aligned}
(d+1)\log(sT+1) &< (d+1)\log\left(\frac{s2^{\frac{s}{2(d+1)}}}{4(d+1)}+1\right)\\
&\leq (d+1)\log\left(2^{\frac{s}{2(d+1)}}\left(\frac{s}{4(d+1)}+1\right)\right)\\
&= (d+1)\left(\frac{s}{2(d+1)}+\log\left(\frac{s}{4(d+1)}+1\right)\right)\\
&\leq \frac{s}{2}+\frac{s}{4\ln(2)} < s,
\end{aligned}
$$

where the penultimate inequality follows since $\log(x) \leq (x-1)/\ln(2)$ for all $x \geq 1$.

We obtain that $2^s > 2^{(d+1)\log(sT+1)}$. Hence, by the pigeonhole principle there are two distinct subsets $A', B' \subseteq \operatorname{supp}(x)$ such that $Mx^{A'} = Mx^{B'}$. We can now define $A = A' \setminus B'$ and $B = B' \setminus A'$ and obtain $Mx^A = Mx^B$. It remains to show that $A, B \neq \emptyset$. Notice that if $A = \emptyset$ then $A' \subseteq B'$, and the last equality of $Mx^{A'} = Mx^{B'}$ implies that $|A'| = |B'|$. This is a contradiction since then $A' = B'$. We conclude that $A \neq \emptyset$. The proof that $B \neq \emptyset$ is analogous. □

Lemma 4.3 (Eisenbrand and Shmonin [ES06])**.** *If [conf-IP] is feasible, then there exists a feasible solution x such that $|\operatorname{supp}(x)| \leq 2(d+1)\log(4(d+1)T)$.*

Proof. Let x be a solution to [conf-IP] that minimizes $|\operatorname{supp}(x)| = s$. Assume by contradiction that $s > 2(d+1)\log(4(d+1)T)$. We show that we can find another solution x' to [conf-IP] with $|\operatorname{supp}(x')| < |\operatorname{supp}(x)|$, contradicting the minimality of $|\operatorname{supp}(x)|$. By Lemma 4.2, there exist two disjoint subsets $A, B \in \operatorname{supp}(x)$ such that $Mx^A = Mx^B$. Moreover, let $\lambda = \min\{x_c : c \in A\}$. Vector $x' := x - \lambda x^A + \lambda x^B$ is also a solution to [conf-IP] and has a strictly smaller support since a configuration $c^* \in \arg\min\{x_c : c \in A\}$ satisfies $x'_{c^*} = 0$. □

4.3 Structural Results

Recall that we call a configuration c simple if $|\operatorname{supp}(c)| \leq \log(T+1)$ and complex otherwise. An important observation to show Theorem 4.1 is that if c is a complex configuration, then $2c$ can be written as the sum of two configurations of smaller support. This is shown by the following Sparsification Lemma.

Lemma 4.4 (Sparsification Lemma)**.** *Let $c \in Q$ be a complex configuration. Then there exist two configurations $c_1, c_2 \in Q$ such that*

1. $\pi \cdot c_1 = \pi \cdot c_2 = \pi \cdot c$,
2. $2c = c_1 + c_2$,
3. $\operatorname{supp}(c_1) \subsetneq \operatorname{supp}(c)$ *and* $\operatorname{supp}(c_2) \subsetneq \operatorname{supp}(c)$.

Proof. Consider for each subset $S \subseteq \operatorname{supp}(c)$, a configuration $c^S \in Q$ such that $c^S_i = c_i$ if $i \in S$ and $c^S = 0$ otherwise. As the number of subsets of $\operatorname{supp}(c)$ is $2^{|\operatorname{supp}(c)|}$, and $c^R \neq c^S$ if and only if $R \neq S$, the collection of vectors $V := \{c^S : S \subseteq \operatorname{supp}(c)\}$ has cardinality $|V| = 2^{|\operatorname{supp}(c)|}$.

On the other hand, for any vector $c^S \in V$ it holds that $\pi \cdot c^S \leq \pi \cdot c \leq T$. Hence, $\pi \cdot c^S \in \{0, 1 \ldots, T\}$ can take only $T+1$ different values. Using that c is a complex configuration and hence $2^{|\operatorname{supp}(c)|} > 2^{\log(T+1)} = T+1$, the pigeonhole principle ensures that there are two different non-empty configurations $c^S, c^R \subseteq V$ with $\pi \cdot c^S = \pi \cdot c^R$. By removing the intersection, we can assume w.l.o.g. that S and R have no intersection. We define $c_1 = c - c^S + c^R$ and $c_2 = c - c^R + c^S$, which satisfy the properties of the lemma as

$$\pi \cdot c_1 = \pi \cdot c - \pi \cdot c^S + \pi \cdot c^R = \pi \cdot c \quad \text{and}$$
$$2c = c - c^S + c^R + c - c^R + c^S = c_1 + c_2.$$

Since $\operatorname{supp}(c_1) \subseteq \operatorname{supp}(c) \setminus S$ and $\operatorname{supp}(c_2) \subseteq \operatorname{supp}(c) \setminus R$, property 3 is satisfied. □

With Lemma 4.4 we are ready to show Theorem 4.1. For the proof it is tempting to apply the lemma iteratively, replacing any complex configuration that is used twice by two configurations with smaller support. This can be repeated until there is no complex configuration taken multiple times. Then we can apply the technique of Lemma 4.3 to the obtained solution to bound the cardinality of the support. However, the last step might break the structure obtained if the solution implied by Lemma 4.3 uses a complex configuration more than once. In order to avoid this issue we consider a potential function. We show that a vector minimizing the chosen potential uses each complex configuration at most once, and that the number of complex configurations in the support is bounded. Finally, we apply the techniques from Lemma 4.3 restricted to variables corresponding to simple configurations.

Proof of Theorem 4.1. Consider the following potential function of a solution $x \in \mathbb{Z}^Q_{\geq 0}$ of [conf-IP],

$$\Phi(x) = \sum_{\text{complex config. } c} x_c |\operatorname{supp}(c)|.$$

Let x be a solution of [conf-IP] with minimum potential $\Phi(x)$, which is well defined since the set of feasible solutions has finite cardinality. We show two properties of x.
P1: $x_c \leq 1$ for each complex configuration $c \in Q$.

Assume otherwise. Consider the two configurations c_1 and c_2 implied by the previous lemma. We define a new solution $x'_e = x_e$ for $e \notin \{c, c_1, c_2\}$, $x'_{c_1} = x_{c_1} + 1$, $x'_{c_2} = x_{c_2} + 1$ and $x'_c = x_c - 2$. Since $|\operatorname{supp}(c_1)| < |\operatorname{supp}(c)|$ and $|\operatorname{supp}(c_2)| < |\operatorname{supp}(c)|$, we obtain that $\Phi(x') < \Phi(x)$ which contradicts the minimality of $\Phi(x)$.
P2: The number of complex configurations in $\operatorname{supp}(x)$ is at most $2(d+1)\log(4(d+1)T)$.

Let $\tilde{x}$ be the vector defined as $\tilde{x}_c = x_c$ if $c \in Q$ is complex, and $\tilde{x} = 0$ if $c \in Q$ is simple. Then Lemma 4.2 implies that there are exist two disjoint subsets $A, B \subseteq \operatorname{supp}(\tilde{x})$ of complex configurations such that $Mx^A = Mx^B$. Thus, the solution $x' = x - x^A + x^B$ and the solution $x'' = x - x^B + x^A$ are feasible for [config-IP]. By linearity, the potential function on the new solutions are $\Phi(x') = \Phi(x) - \Phi(x^A) + \Phi(x^B)$ or respectively $\Phi(x'') = \Phi(x) - \Phi(x^B) + \Phi(x^A)$. If $\Phi(x^A) > \Phi(x^B)$ or $\Phi(x^B) > \Phi(x^A)$ then we have constructed a new solution with smaller potential, contradicting our assumption on the minimality of $\Phi(x)$. We conclude that $\Phi(x^B) = \Phi(x^A)$ and thus $\Phi(x) = \Phi(x')$. By construction of x', we obtain that $x'_c > x_c \geq 1$ for any complex configuration $c \in B$. Having multiplicity ≥ 2 for a complex configuration c, we can proceed as in Case 1 to find a new solution with decreased potential, which yields a contradiction.

Given these two properties, to conclude the theorem it suffices to upper bound the number of simple configurations by $2(d+1)\log(4(d+1)T)$. Suppose this property is violated, then we find two sets $A, B \subseteq \text{supp}(x)$ of simple configurations (see Lemma 4.2) with $Mx^A = Mx^B$ and proceed as in Lemma 4.3. Since Lemma 4.3 is only applied to simple configurations, properties P1 and P2 continue to hold and the theorem follows. □

Our techniques, in particular our Sparsification Lemma, imply two corollaries on the structure of the knapsack polytope and the LP-relaxation implied by the [conf-IP].

Corollary 4.5. *Every vertex of conv.hull(Q) is a simple configuration. Moreover, the total number of simple configurations in Q is upper bounded by $2^{O(\log^2(T)+\log^2(d))}$ and thus the same expression upper bounds the number of vertices of conv.hull(Q).*

Proof. Consider a complex configuration $c \in Q$. By Lemma 4.4 we know that there exist $c_1, c_2 \in Q$ with $c_1, c_2 \neq c$ such that $2c = c_1 + c_2$. Hence, c is not a vertex of Q as it can be written as a convex combination $c = c_1/2 + c_2/2$.

To bound the number of simple configurations fix a set $D \subseteq \{1, \ldots, d\}$. Notice that the number of configurations c with $\text{supp}(c) = D$ is at most $T^{|D|}$. For simple configurations it suffices to take D with cardinality at most $\log(T+1)$. Since the number of subsets $D \subseteq \{1, \ldots, d\}$ with cardinality i is $\binom{d}{i}$, we obtain that the number of simple configurations is at most

$$\sum_{i=0}^{\lfloor \log(T+1) \rfloor} \binom{d}{i} \times (T+1)^{\log(T+1)} \leq (\log(T+1)+1) d^{\log(T+1)} \times (T+1)^{\log(T+1)}$$
$$= 2^{\log(\log(T+1)+1)+\log(d)\log(T+1)} \times 2^{\log(T+1)\log(T+1)} = 2^{O(\log^2(d)+\log^2(T))}.$$ □

The following corollary follows as each complex configuration can be represented by a convex combination of simple configurations.

Corollary 4.6. *Let [conf-LP] be the LP relaxation of [conf-IP], obtained by changing each constraint $x_c \in \mathbb{Z}_{\geq 0}$ to $x_c \geq 0$ for all $c \in Q$. If the LP is feasible then there exists a solution x such that each configuration $c \in \text{supp}(x)$ is simple.*

Proof. Consider a solution x of [conf-LP]. Assume that there exists $c \in Q$ such that c is complex and $x_c > 0$. Then by the previous corollary, configuration c can be written as $c = \sum_{q \in Q} \lambda_q q$, where $\sum_{q \in Q} \lambda_q = 1$, $\lambda_q \geq 0$ for all $q \in Q$, and $\lambda_q = 0$ if $q \in Q$ is complex. Consider a new solution x' defined as

$$x'_q = \begin{cases} 0 & \text{if } q = c, \\ x_q + \lambda_q \cdot x_c & \text{if } q \neq c. \end{cases}$$

This new solution is also feasible for [conf-LP]. As $x'_c = 0$, the number of complex configurations in the support of the solution is reduced by 1. This procedure can be repeated until we have a solution $\hat{x}$ whose support contains only simple configurations. □

4.4 Applications to Scheduling on Parallel Machines

In what follows we show how to exploit the structural insights of the previous section to derive faster algorithms for parallel machines scheduling problems. We start by considering $P||C_{\max}$, where we seek to assign a set of jobs $\mathcal{J}$ with processing times $p_j \in \mathbb{Z}_{>0}$ to a set

$\mathcal{M}$ of m machines. For a given assignment $a : \mathcal{J} \mapsto \mathcal{M}$, we define the load of a machine i as $\sum_{j:a(j)=i} p_j$ and the *makespan* as the maximum load of jobs over all machines, which is the minimum time needed to complete the execution of all jobs on the processors. The goal is to find an assignment $\mathcal{J} \mapsto \mathcal{M}$ that minimizes the makespan.

We first follow well known rounding techniques [Alo+97; Alo+98; HS87; Hoc97]. Consider an error tolerance $0 < \varepsilon < 1/3$ such that $1/\varepsilon^2$ is an integer. To get an estimation of the optimal makespan, we follow the standard dual approximation approach. First, we can use, e.g., the 2-approximation algorithm by Graham [Gra66] to get an initial guess of the optimal makespan. Using binary search, we can then estimate the optimal makespan within a factor of $(1 + \varepsilon)$ in $O(\log(1/\varepsilon))$ iterations. Therefore, it remains to give an algorithm that decides for a given makespan T, if there exists an assignment with makespan $(1 + O(\varepsilon))T$ or reports that there exists no assignment with makespan $\leq T$.

For a given makespan T we define the set of big jobs $\mathcal{J}_{\text{big}} = \{j \in \mathcal{J} : p_j \geq \varepsilon T\}$ and the set of small jobs $\mathcal{J}_{\text{small}} = \mathcal{J} \setminus \mathcal{J}_{\text{big}}$. The following lemma shows that small jobs can be replaced from the instance by adding big jobs, each of size εT, as placeholders. Let S be the sum of processing times of jobs in $\mathcal{J}_{\text{small}}$ and let S^* denote the next value of S rounded up to the next multiple of εT, that is, $S^* = \varepsilon T \cdot \lceil S/(\varepsilon T) \rceil$. We define a new instance containing only big jobs by $\mathcal{J}^* = \mathcal{J}_{\text{big}} \cup \mathcal{J}_{\text{new}}$, where $\mathcal{J}_{\text{new}}$ contains $S^*/(\varepsilon T) \in \mathbb{N}$ jobs of size εT.

Lemma 4.7. *Given a feasible assignment $a : \mathcal{J} \mapsto \mathcal{M}$ of jobs with makespan T. Then there exists a feasible assignment $a_B : \mathcal{J}^* \mapsto \mathcal{M}$ of makespan $T^* \leq (1 + \varepsilon)T$. Similarly, an assignment of jobs in $\mathcal{J}^*$ of makespan T^* can be transformed to an assignment of $\mathcal{J}$ of makespan at most $(1 + \varepsilon)T^*$.*

Proof. We modify the assignment a of jobs in $\mathcal{J}$ by replacing the set of small jobs on each machine by jobs in $\mathcal{J}_{\text{new}}$. Let S_i be the total processing time of small jobs assigned to machine i. Then the small jobs are replaced by (at most) $S_i^*/(\varepsilon T)$ jobs in $\mathcal{J}_{\text{new}}$, where S_i^* denotes the value of S_i rounded up to the next multiple of εT. As $\sum \frac{S_i^*}{\varepsilon T} \geq \lfloor \sum \frac{S_i}{\varepsilon T} \rfloor = \frac{S^*}{\varepsilon T}$, the new solution processes all jobs in $\mathcal{J}_{\text{new}}$ and the load on each machine increases hence by at most εT. Having an assignment for the big jobs $\mathcal{J}^*$, we can easily obtain a schedule for jobs $\mathcal{J}$, by adding the small items greedily into the space of the placeholder jobs $\mathcal{J}_{\text{new}}$. □

By scaling the processing times of jobs in $\mathcal{J}^*$, we can assume that the makespan T has value $1/\varepsilon^2$. Also notice that we can assume that $p_j \leq T$ for all j, otherwise we cannot pack all jobs within makespan T. This implies that each job $j \in \mathcal{J}^*$ has a processing time of $1/\varepsilon \leq p_j \leq 1/\varepsilon^2$. In the following we give a transformation of big jobs in $\mathcal{J}^*$ by rounding their processing times. We first round the jobs to the next power of $1 + \varepsilon$ as $p'_j = (1 + \varepsilon)^{\lceil \log_{(1+\varepsilon)} p_j \rceil}$, and thus all rounded processing times belong to $\Pi' = \{(1 + \varepsilon)^k : 1/\varepsilon \leq (1 + \varepsilon)^k \leq (1 + \varepsilon)/\varepsilon^2 \text{ and } k \in \mathbb{N}\}$. We further round processing times p'_j to the next integer $\bar{p}_j = \lceil p'_j \rceil$ and define a new set $\Pi = \{\lceil p \rceil : p \in \Pi'\}$. Notice that Π only contains integers and $|\Pi| \leq |\Pi'| \in O((1/\varepsilon)\log(1/\varepsilon))$.

Lemma 4.8. *If there is a feasible schedule of jobs $\mathcal{J}^*$ with processing times p_j onto m machines with makespan $T^* \leq (1 + \varepsilon)T$, then there is also a feasible schedule of jobs $\mathcal{J}^*$ with rounded processing $\bar{p}_j$ with a makespan of at most $(1 + 5\varepsilon)T$. Furthermore, the number of different processing times is at most $|\Pi| \in O((1/\varepsilon)\log(1/\varepsilon))$.*

Proof. Consider a feasible schedule of jobs in $\mathcal{J}^*$ with processing times p_j onto m machines with makespan T^*. Let $J_{i_1}, \ldots, J_{i_r}$ be the set of jobs processed on machine i i.e. $a(J_{i_k}) = i$

for $k = 1, \ldots, r$. Then $\sum_{j=1}^{r} p'_j \leq \sum_{j=1}^{r}(1+\varepsilon)p_j \leq (1+\varepsilon)T^*$. Hence, the same assignment a with processing times p'_j yields a makespan of at most $(1+\varepsilon)T^* \leq (1+\varepsilon)^2 T = 1/\varepsilon^2 + 2/\varepsilon + 1$. Since $p'_j \geq p_j \geq 1/\varepsilon$, on every machine are at most $1/\varepsilon + 2$ jobs. Hence, rounding the processing times p'_j of each job to the next integer increase the load on each machine by at most $1/\varepsilon + 2$. Recalling that $\varepsilon < 1/3$, we obtain a feasible schedule with makespan at most $(1+\varepsilon)T^* + 1/\varepsilon + 2 \leq 1/\varepsilon^2 + 3/\varepsilon + 3 < T + 5\varepsilon T$. □

In what follows we give an algorithm that decides in polynomial time the existence of a solution for instance $\mathcal{J}^*$ with processing times $\bar{p}_j$ and makespan $\bar{T} = \lfloor (1+5\varepsilon)T \rfloor$. We call numbers in Π by $\pi_1, \ldots, \pi_d$ and define the vector $\pi = (\pi_1, \pi_2, \ldots, \pi_d) \in \mathbb{N}^d$ of rounded processing times. We consider *configurations* to be vectors in $Q = \mathcal{P} \cap \mathbb{Z}^d$, where $\mathcal{P} = \{c \in \mathbb{R}^d_{\geq 0} : \pi \cdot c \leq \bar{T}\}$ is a knapsack polytope (see Section 4.3). As before, we say that a configuration is simple if $|\operatorname{supp}(c)| \leq \log(\bar{T}+1)$, and complex otherwise. For a given assignment of jobs to machines, we say that a machine follows a configuration c if c_k is the number of jobs of size π_k assigned to the machine. We denote by $Q_c \subseteq Q$ the set of complex configurations and by $Q_s \subseteq Q$ the set of simple configurations.

Let b_k be the number of jobs of size π_k in the instance $\mathcal{J}^*$ (with processing times $\bar{p}$). Consider an ILP with integer variables x_c for each $c \in Q$, which denote the number of machines that follow configuration c. With these parameters the problem of scheduling all jobs in a solution of makespan $\bar{T}$ is equivalent to finding a solution to [conf-IP]. To solve the ILP we use, among other techniques, Kannan's algorithm [Kan87] which is an improvement on the algorithm by Lenstra [Len83]. The algorithm has a running time of $2^{O(N \log N)} s$ where N is the number of variables and s is number of bits used to encode the input of the ILP in binary.

By Theorem 4.1, if [conf-IP] is feasible then there exists a thin solution. In particular if one configuration c is used by more than one machine then c is simple, and the total number of used configurations is $4(d+1)\log(4(d+1)\bar{T}) \in O((1/\varepsilon)\log^2(1/\varepsilon))$. Additionally, the number of machines following a complex configurations is at most $2(d+1)\log(4(d+1)\bar{T}) \in O((1/\varepsilon)\log^2(1/\varepsilon))$. We consider the following strategy to decide the existence of a schedule of makespan $\bar{T}$.

Algorithm 4.9.

1. *For each processing time π_k, guess the number $b^c_k \leq b_k$ of jobs covered by complex configurations.*

2. *Find a minimum number of machines m^c to schedule jobs b^c with makespan $\bar{T}$.*

3. *Guess the support of simple configurations $\bar{Q}_s \subseteq Q_s$ used by a thin solution, with $|\bar{Q}_s| \leq 4(d+1)\log(4(d+1)\bar{T}) \in O((1/\varepsilon)\log^2(1/\varepsilon))$.*

4. *Solve the ILP restricted to configurations in $\bar{Q}_s$:*

$$\begin{aligned} &\sum_{c \in \bar{Q}_s} c \cdot x_c = b - b^c, \\ &\sum_{c \in \bar{Q}_s} x_c = m - m^c, \\ &x_c \in \mathbb{Z}_{\geq 0} \qquad\qquad \text{for all } c \in \bar{Q}_s. \end{aligned}$$

One of the key observations to prove the running time of the algorithm is that the number of simple configurations $|Q_s|$ is bounded by a quasi polynomial term:

$$|Q_s| \leq 2^{O(\log^2(1/\varepsilon))}.$$

This follows easily by Corollary 4.5, using that $|\bar{T}| \in O(1/\varepsilon^2)$ and $d = |\Pi| \in O((1/\varepsilon)\log(1/\varepsilon))$.

Lemma 4.10. *Algorithm 4.9 can be implemented with a running time of* $2^{O((1/\varepsilon)\log^4(1/\varepsilon))}\log(n)$.

Proof. In step 1, the algorithm guesses which jobs are processed on machines following a complex configurations. Since each configuration contains at most $O(1/\varepsilon)$ jobs, there are at most $O(m^c/\varepsilon) = O((1/\varepsilon^2)\log^2(1/\varepsilon))$ jobs assigned to such machines. For each size $\pi_k \in \Pi$, we guess the number b_k^c of jobs of size π_k assigned to such machines. Hence, we can enumerate all possibilities for jobs assigned to complex machines in time $2^{O((1/\varepsilon)\log^2(1/\varepsilon))}$. After guessing the jobs, we can assign them to a minimum number of machines in step 2 (with makespan $\bar{T}$) with a simple dynamic program that stores vectors $(\ell, z_1, \ldots, z_d)$ with $z_k \leq b_k^c$ being the number of jobs of size π_k used in the first $\ell \leq m^c$ processors [JR11]. The size of the dynamic programming table is $O(m^c \prod_{k=1}^{d}(b_k^c+1))$. For any vector $(\ell, z_1, \ldots, z_d)$, determining whether it corresponds to a feasible solution can be done by checking all vectors of the type $(\ell-1, z_1', \ldots, z_d')$ for $z_k' \leq z_k$. Thus, the running time of the dynamic program is $O(m^c[\prod_{k=1}^{d}(b_k^c+1)]^2)$. Since $b_k^c \in O((1/\varepsilon^2)\log^2(1/\varepsilon))$ for each k, recalling that $m^c \in O((1/\varepsilon)\log^2(1/\varepsilon))$, and that $d = |\Pi| \in O((1/\varepsilon)\log(1/\varepsilon))$, we obtain that step 2 can be implemented with $2^{O((1/\varepsilon)\log^2(1/\varepsilon))}$ running time.

In step 3, our algorithm guesses the support of a thin solution x. Recall that if x is thin then $|\operatorname{supp}(x)| \leq 4(d+1)\log(4(d+1)\bar{T}) = O((1/\varepsilon)\log^2(1/\varepsilon))$. Let $D = 4(d+1)\log(4(d+1)\bar{T})$. Then this guess can be done in time

$$\sum_{i=0}^{D}\binom{|Q_s|}{i} \leq (D+1)|Q_s|^D \leq 2^{O((1/\varepsilon)\log^4(1/\varepsilon))}.$$

We remark that for this step is that thin solutions are particularly useful. Indeed, guessing the support on the original ILP takes time $2^{O((1/\varepsilon)^2\log^3(1/\varepsilon))}$.

In step 4, the number of variables of the restricted ILP is $4(d+1)\log(4(d+1)\bar{T}) = O((1/\varepsilon)\log^2(1/\varepsilon))$. Moreover, the size of the input is bounded by $O((1/\varepsilon^2)\log^3(1/\varepsilon)\log(n))$. Running Kannan's algorithm [Kan87] to solve the ILP takes time $2^{O((1/\varepsilon)\log^3(1/\varepsilon))}\log(n)$. Hence, the total running time of our algorithm can be bounded by $2^{O((1/\varepsilon)\log^4(1/\varepsilon))}\log(n)$. □

Putting all pieces together, we conclude with the following theorem.

Theorem 4.11. *The minimum makespan problem on parallel machines* $P||C_{\max}$ *admits an EPTAS with running time* $2^{O((1/\varepsilon)\log^4(1/\varepsilon))} + O(n\log n)$.

Proof. Consider a scheduling instance with job set $\mathcal{J}$, processing times p_j for $j \in \mathcal{J}$ and machine set $\mathcal{M}$. The greedy algorithm by Graham to obtain a 2-approximation can be implemented in $O(n\log n)$. After guessing the makespan T, the processing times are sorted and rounded as described in Lemma 4.8. The rounding step can easily be implemented in $O(n)$ time. Applying Algorithm 4.9 after the rounding needs, according to Theorem 4.10, a running time of $2^{O((1/\varepsilon)\log^4(1/\varepsilon))}\log(n)$ time. Since there are at most $O(\log(1/\varepsilon))$ many guessing rounds for the makespan, we obtain a total running time of $O(n\log n + \log(1/\varepsilon)\cdot n) + 2^{O((1/\varepsilon)\log^4(1/\varepsilon))}\log(n)$.

If $n \leq 2^{\frac{1}{\varepsilon}\log^4(\frac{1}{\varepsilon})}$ then the running time is upper bounded by $2^{O(\frac{1}{\varepsilon}\log^4(\frac{1}{\varepsilon}))}$, otherwise, the running time is at most $O(n \log n)$. In any case, the running time can be bounded by $2^{O(\frac{1}{\varepsilon}\log^4(\frac{1}{\varepsilon}))} + O(n \log n)$. □

4.4.1 Extension to other objectives

We now consider a more general family of objective functions defined by Alon et al. [Alo+97; Alo+98]. For a fixed function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, we consider the following two objective functions:

$$\text{(I) } \min \sum_{i \in \mathcal{M}} f(\ell_i) \qquad \text{(II) } \min \max_{i \in \mathcal{M}} f(\ell_i),$$

where ℓ_i denotes the load of machine i. Analogously, we study maximization versions of the problems

$$\text{(I')} \max \sum_{i \in \mathcal{M}} f(\ell_i) \qquad \text{(II')} \max \min_{i \in \mathcal{M}} f(\ell_i),$$

For the minimization versions of the problem we assume that f is convex, while for (I') and (II') we assume it is concave. Moreover, we will need that the function satisfies the following sensitivity condition.

Condition 4.12. *For all $\varepsilon > 0$ there exists $\delta = \delta(\varepsilon) > 0$ such that for all $x, y \in \mathbb{R}_{\geq 0}$,*

$$(1-\delta)y \leq x \leq (1+\delta)y \quad \Rightarrow \quad (1-\varepsilon)f(y) \leq f(x) \leq (1+\varepsilon)f(y).$$

Alon et al. showed that each problem in that family admits a PTAS with running time $h(\varepsilon) + O(n \log n)$, where $h(\varepsilon)$ is a constant term that depends only on ε. Moreover, if $\delta(\varepsilon)$ in the condition further satisfies that $1/(\delta(\varepsilon)) \in O(1/\varepsilon)$, the running time is $2^{(1/\varepsilon)^{\text{poly}(1/\varepsilon)}} + O(n \log n)$. In what follows we show how to improve this dependency. Since $1/(\delta(\varepsilon)) \in O(1/\varepsilon)$, we know that, for small enough ε, there exists a constant γ (independent of ε and δ) such that $1/\delta \leq \gamma/\varepsilon$. Moreover, we can assume w.l.o.g. that $\delta \leq \varepsilon$, and thus $\delta \leq \varepsilon \leq \gamma\delta$.

It is worth noticing that many interesting functions belong to this family. In particular (II) with $f(x) = x$ corresponds to the minimum makespan problem, (I) with $f(x) = x^p$, for constant p, corresponds to a problem that is equivalent to minimizing the L_p-norm of the vector of loads. Similarly, (II') with $f(x) = x$ corresponds to maximizing the minimum machine load. Notice that for all those objectives we have that $1/\delta = O(1/\varepsilon)$.

The techniques of Alon et al. [Alo+98] are based on a rounding method and then solving an ILP. We based our results in the same rounding techniques. Consider an arbitrary instance of a scheduling problem on identical machines with objective function (I), (II), (I') or (II'). Their first observation is that, if $L = \sum_j p_j/m$ is the average machine load, then a job with $p_j \geq L$ is scheduled alone on a machine in an optimal solution [Alo+98]. Hence, we can remove such job and a machine from the instance. In what follows, we assume without loss of generality, that $p_j < L$ for all j. For the sake of brevity, we summarize the rounding techniques of Alon et al. in the following theorem.

Theorem 4.13 (Alon et al. [Alo+98])**.** *Consider an instance for the scheduling problem with job set $\mathcal{J}$, identical machines $\mathcal{M}$, and processing times p_j for $j \in \mathcal{J}$ such that $p_j < L$ for all j. There exists a linear time algorithm that creates a new instance I' with job set $\mathcal{J}'$, machine set $\mathcal{M}$, and processing times p'_j. Moreover, there is an integer $\lambda \geq 1/\delta$ with $\lambda \in O(1/\delta)$ such that the new instance satisfies the following:*

1. *Each job j in I' has processing time $L/\lambda \le p'_j \le L$, and p'_j is a integer multiple of L/λ^2.*

2. *If $L' = \sum_j p'_j/m$ then $L \le L' \le (1+2/\lambda)L$.*

3. *Let* Opt *and* Opt$'$ *be the optimal value of instances I and I', respectively. Then $(1-\varepsilon)\text{Opt} \le \text{Opt}' \le (1+\varepsilon)\text{Opt}$.*

4. *There exists a linear time algorithm that transforms a feasible solution for instance I' with objective value V to a feasible solution for I with objective value V' such that $(1-\varepsilon)V \le V' \le (1+\varepsilon)V$.*

Given this result, it suffices to find a $(1+\varepsilon)$-approximate solution for instance I'. To do so, we further round the processing times as in the previous section by defining $\bar{p}_j$ as the value $(1+\delta)^{\lceil \log_{1+\delta} p'_j \rceil}$ rounded up to the next multiple of L/λ^2 for all $j \in \mathcal{J}'$. Notice that $\bar{p}_j \le (1+\delta)^{\lceil \log_{1+\delta} p'_j \rceil} + L/\lambda^2 \le (1+\delta)p'_j + L/\lambda^2 \le (1+\delta)p'_j + p'_j/\lambda \le (1+2\delta)p'_j \le (1+\delta)^2 p'_j$. Hence, for any assignment that gives a load ℓ_i on machine i for p'_j, the same assignment has a load $\bar{\ell}_i$ with $\ell_i \le \bar{\ell}_i \le (1+\delta)^2\ell_i$. By Condition 4.12 we conclude that the new optimal value $\overline{\text{Opt}}$ satisfies that $(1-O(\varepsilon))\text{Opt} \le \overline{\text{Opt}} \le (1+O(\varepsilon))\text{Opt}$.

Let $\Pi = \{\pi_1, \dots, \pi_d\}$ be the distinct values that the processing times $\bar{p}_j$ can take. Notice that $d = |\Pi| = O((1/\delta)\log(1/\delta))$. We consider the knapsack polytope with capacity $\bar{T} := 4L$, that is $\mathcal{P} = \{c \in \mathbb{R}^d_{\ge 0} : \pi \cdot c \le \bar{T}\}$. Notice that π and $\bar{T}$ are integer multiples of L/λ^2, and thus $\mathcal{P} = \{c \in \mathbb{R}^d_{\ge 0} : \pi/(L/\lambda^2) \cdot c \le \bar{T}/(L/\lambda^2)\}$. The following lemma, that is a simple adaptation of an observation by Alon et al. [Alo+98], shows that there exists an optimal solution for the rounded instance that uses only configurations in $\mathcal{P}$.

Lemma 4.14. *For $\varepsilon > 0$ small enough, the rounded instance with processing times $\bar{p}_j$ admits an optimal solution with makespan at most $4L$.*

Proof. Among all optimal solutions to the problem, consider one that minimizes $\sum_i \ell_i^2$, where ℓ_i is the load on machine i. Assume that there exists a machine i such that $\ell_i > 4L$. Notice that

$$\sum_j \bar{p}_j/m \le (1+\delta)^2 \sum_j p'_j/m = (1+\delta)^2 L' \overset{\text{Theorem 4.13}}{\le} (1+\delta)^2(1+2/\lambda)L \le (1+\delta)^4 L.$$

Since $\delta \le \varepsilon$, for ε small enough ($\varepsilon \le 1/10$ suffices) we have that $(1+\delta)^4 \le 2$ and thus $\sum_j \bar{p}_j/m \le 2L$. Also, recall that $\bar{p}_j \le (1+\delta)^2 p'_j \le (1+\delta)^2 L \le 2L$ for any j, where the second to last inequality follows from Theorem 4.13. Since $\ell_{\min} = \min_i \ell_i \le \sum_j \bar{p}_j/m \le 2L$, then $\ell_i - \ell_{\min} > 4L - 2L = 2L$. Then, for any job j, we have that $p_j < \ell_i - \ell_{\min}$. Let j^* be any job assigned to machine i. Hence, in particular we have that $p_{j^*} < \ell_i - \ell_{\min}$.

Recall that for problems (I) and (II) function f is convex. Hence, it holds that $f(x+\Delta) + f(y-\Delta) \le f(x) + f(y)$ for all $0 \le x \le y$ with $0 \le \Delta \le y - x$ [Alo+98]. Moreover, the inequality becomes strict if f is strictly convex. Setting $x = \ell_{\min}$, $y = \ell_i$ and $\Delta = p_{j^*}$, the inequality implies that moving job j^* to machine $i^* \in \arg\min_i \ell_i$ decreases strictly $\sum_i \ell_i^2$. Moreover, the objective function (I) does not increase when performing this move, which yields a contradiction for this objective. Similarly, for problem (II) the objective does not increase since $\max\{f(\xi) : \xi \in [x, y]\}$ is always attained at x or y for f convex. This yields a contradiction for (II).

Analogously, for problems (I') and (II') function f is concave and thus $f(x+\Delta) + f(y-\Delta) \ge f(x) + f(y)$ holds for all $0 \le x \le y$ with $0 \le \Delta \le y - x$ [Alo+98]. Hence,

moving job j^* to machine i^* decreases $\sum_i \ell_i^2$ but does not increase the objective (I'). Since f concave implies that $\min\{f(\xi) : \xi \in [x, y]\}$ is always attained at x or y, we also obtain a contradiction for (II'). The lemma follows. □

Let $L = \sum_j p_j/m$ be the average machine load (of the original instance). After our rounding we obtain an instance I' with job set $\mathcal{J}'$ and processing times $\bar{p}_j$ for $j \in \mathcal{J}'$. Moreover, the $\bar{p}_j$ are multiples of L/λ^2, where $\lambda \geq 1/\delta$ is an integer such that $\lambda = O(1/\delta)$, and also $\bar{p}_j \geq L/\lambda$. It holds that there exists an optimal solution of the rounded instance with makespan at most $4L$, see Lemma 4.14 (in particular $\bar{p}_j \leq 4L$ for all j). Let $\Pi = \{\pi_1, \ldots, \pi_d\}$ be the distinct values that the processing times $\bar{p}_j$ can take. Our rounding guarantees that $d = |\Pi| = O((1/\delta)\log(1/\delta))$. We consider the knapsack polytope with capacity $\bar{T} := 4L$, that is $\mathcal{P} = \{c \in \mathbb{R}^d_{\geq 0} : \pi \cdot c \leq \bar{T}\}$. Notice that π and $\bar{T}$ are integer multiples of L/λ^2, and that $\mathcal{P}$ can also be written as $\{c \in \mathbb{R}^d_{\geq 0} : \pi/(L/\lambda^2) \cdot c \leq \bar{T}/(L/\lambda^2)\}$.

As before, we say that a configuration is simple if $|\operatorname{supp}(c)| \leq \log(\bar{T} + 1)$, and complex otherwise. We denote by $Q_c \subseteq Q$ the set of complex configurations and by $Q_s \subseteq Q$ the set of simple configurations. In what follows we focus on objective function (I).

We set an ILP for the problem as before. Notice that each configuration c incurs a cost of $f_c := f(\pi \cdot c)$. Moreover, we round and scale the values f_c by defining $\bar{f}_c = \lceil f_c/(\varepsilon f_{\min})\rceil$, where $f_{\min} = \min_{c \in Q} f_c$. It is not hard to see that solving a problem with those coefficients yields a $(1+\varepsilon)$-approximate solution to the optimal solution of I' with processing times $\bar{p}_j$. Let also b_k be the number of jobs j of processing time $\bar{p}_j = \pi_k$ in $\mathcal{J}'$. Consider the ILP obtained by adding to [conf-IP] the objective function $\min \sum_{c \in Q} \bar{f}_c \cdot x_c$. We call this ILP [cost-conf-IP]. With our previous discussion, it suffices to solve this ILP optimally. To solve this problem, we first notice that the largest coefficient in the objective can be bounded as follows.

Lemma 4.15. *If f satisfies Condition 4.12 then the largest value $\max_{c \in Q} \bar{f}_c$ is upper bounded by $1/\delta^{O(1)}$.*

Proof. We first bound $(\max_{c \in Q} f_c)/(\varepsilon f_{\min})$. Notice that Condition 4.12 implies that f is continuous on $\mathbb{R}_{\geq 0}$, and thus it admits a minimum and maximum in the interval $[L/\lambda, 4L]$. Let $x_{\min} \in \arg\min\{f(x) : x \in [L/\lambda, 4L]\}$ and $x_{\max} \in \arg\max\{f(x) : x \in [L/\lambda, 4L]\}$.

Consider first the case in which $x_{\min} \leq x_{\max}$ (this is not always true since f might not be monotone). We now use Condition 4.12 iteratively. Let $y^k := (1+\delta)^k x_{\min}$. Since $y^k \leq y^{k-1}(1+\delta)$, Condition 4.12 implies that $f(y^k) \leq (1+\varepsilon) f(y^{k-1})$. Iterating this idea we obtain that $f(y^k) \leq (1+\varepsilon)^k f(y^0)$. Taking $k = \lceil \log_{1+\delta}(x_{\max}/x_{\min})\rceil$ implies that $x_{\max} \leq y^k \leq x_{\max}(1+\delta)$ and thus, by Condition 4.12, it holds that $f(y^k) \geq (1-\varepsilon) f(x_{\max})$. Recall that $\delta \leq \varepsilon \leq \gamma\delta$. We obtain that

$$
\begin{aligned}
f(x_{\max}) \leq \frac{f(y^k)}{1-\varepsilon} &\leq \frac{(1+\varepsilon)^k}{1-\varepsilon} f(x_{\min}) \\
&\leq \frac{(1+\varepsilon)^{\log_{1+\delta}(x_{\max}/x_{\min})+1}}{1-\varepsilon} f(x_{\min}) \\
&\leq \frac{(1+\varepsilon)^{\log_{1+\delta}(4\lambda)+1}}{1-\varepsilon} f(x_{\min}) \\
&= \frac{1+\varepsilon}{1-\varepsilon} (4\lambda)^{\log_{1+\delta}(1+\varepsilon)} f(x_{\min}) \\
&\leq \frac{1+\varepsilon}{1-\varepsilon} (4\lambda)^{\log_{1+\delta}(1+\gamma\delta)} f(x_{\min}) \\
&= (1/\delta)^{O(1)} f(x_{\min}),
\end{aligned}
$$

where the last expression follows since $\log_{1+\delta}(1+\gamma\delta) = \ln(1+\gamma\delta)/\ln(1+\delta) \leq \gamma\delta/\ln(1+\delta) = O(\gamma) = O(1)$ (for δ small enough), and since $\lambda = O(1/\delta)$. We conclude that

$$
\max_{c \in Q} \bar{f}_c \leq (f(x_{\max}))/(\varepsilon f(x_{\min})) + 1 \leq (1/\varepsilon)(1/\delta)^{O(1)} + 1 = (1/\delta)^{O(1)}.
$$

For the case in which $x_{\max} \leq x_{\min}$ we define the sequence $y^k := (1-\delta)^k x_{\min}$. The rest of the proof is analogous and the details are left to the reader. □

As we now must consider the objective function, we cannot simply apply Theorem 4.1 to [cost-conf-ILP]. However, we can prove a slightly weaker version by decomposing the ILP in several smaller ones and applying the theorem to each of them.

Theorem 4.16. *If [cost-conf-IP] is feasible, then there exists an optimal solution x satisfying:*

1. $\sum_{c \in Q_c} x_c \in O((1/\delta^3)\log^2(1/\delta))$, *and*
2. $|\operatorname{supp}(x) \cap Q_s| \in O((1/\delta)\log^2(1/\delta))$.

Proof. Notice that the load of each configuration $\pi \cdot c$ is a multiple of L/λ^2, and thus $\pi \cdot c \in \{L/\lambda, L/\lambda + L/(\lambda^2), \ldots, 4L\}$. We classify the configurations according to their loads, $Q^\ell := \{c \in Q : \pi \cdot c = L/\lambda + \ell \cdot L/(\lambda^2)\}$, for $\ell \in \{0, \ldots, 4\lambda^2 - \lambda\}$. Let x^* be an optimal solution of [cost-conf-IP]. Then we can considered an ILP for each load value ℓ:

$$
\begin{aligned}
[\text{conf-IP}]_\ell \quad & \sum_{c \in Q^\ell} c \cdot x_c = \sum_{c \in Q^\ell} c \cdot x^*_c, && (4.4) \\
& \sum_{c \in Q^\ell} x_c = \sum_{c \in Q^\ell} x^*_c, && (4.5) \\
& x_c \in \mathbb{Z}_{\geq 0} \qquad \text{for all } c \in Q^\ell. && (4.6)
\end{aligned}
$$

Scaling π by multiplying it by λ^2/L we obtain an integral vector (since π is an integer multiple of $L/(\lambda^2)$), we can apply Theorem 4.1 to each ILP $[\text{conf-IP}]_\ell$, which yields that there exists a thin solution x^ℓ. In particular the number of complex configurations in x^ℓ is $\sum_{c \in Q_c \cap Q^\ell} x^\ell_c \in O((1/\delta)\log^2(1/\delta))$. Since $\bar{f}_c$ depends only on the load of c, concatenating these solutions yields a solution $x' := (x^\ell)_\ell$ that is optimal for [cost-conf-IP], such that

$\sum_{c \in Q_c} x'_c \in O((\lambda^2) \cdot (1/\delta)\log^2(1/\delta)) = O((1/\delta^3)\log^2(1/\delta))$. It remains to bound the number of simple configurations in the support. To this end, we consider the ILP restricted to simple configurations as follows:

$$
\begin{aligned}
[\text{cost-conf-IP}]_s \quad & \min \sum_{c \in Q_s} \bar{f}_c \cdot x_c \\
& \sum_{c \in Q_s} c \cdot x_c = b - \sum_{c \in Q_c} c \cdot x'_c, && (4.7)\\
& \sum_{c \in Q_s} x_c = m - \sum_{c \in Q_c} x'_c, && (4.8)\\
& x_c \in \mathbb{Z}_{\geq 0} \qquad \text{for all } c \in Q_s. && (4.9)
\end{aligned}
$$

We apply the result of Eisenbrand and Shmonin [ES06] to this ILP. In its more general form, this result ensures the existence of a solution x'' with support of size $O(N(\log(N) + \Delta))$, where N is the number of restrictions and Δ is the encoding size of the largest coefficient appearing in the cost vector and restriction matrix. In our case $N = d + 1 = O((1/\delta)\log(1/\delta))$, and $\Delta = O(\log(\max\{1/\delta, \max_{c \in Q} \bar{f}_c\}) = O(\log(1/\delta))$ (Lemma 4.15). Thus $O(N(\log(N) + \Delta)) = O((1/\delta)\log^2(1/\delta))$. The theorem follows by concatenating $(x''_c)_{c \in Q_s}$ with $(x'_c)_{c \in Q_c}$. □

Finally, we use the structure given by the theorem to solve this ILP optimally.

Algorithm 4.17.

1. *For each processing time π_k, guess the number $b^c_k \leq b_k$ of jobs covered by complex configurations.*
2. *Guess the number m^c of machines that schedule jobs b^c.*
3. *Compute an optimal solution for instance with number of jobs b^c on m^c machines with a dynamic program.*
4. *Guess the support of simple configurations $\bar{Q}_s \subseteq Q_s$ used by the solution implied by Theorem 4.1, with $|\bar{Q}_s| \in O((1/\delta)\log^2(1/\delta))$.*
5. *Solve the ILP restricted to configurations in $\bar{Q}_s$:*

$$
\begin{aligned}
& \min \sum_{c \in \bar{Q}_s} \bar{f}_c \cdot x_c \\
& \sum_{c \in \bar{Q}_s} c \cdot x_c = b - b^c, \\
& \sum_{c \in \bar{Q}_s} x_c = m - m^c, \\
& x_c \in \mathbb{Z}_{\geq 0} \qquad \textit{for all } c \in \bar{Q}_s.
\end{aligned}
$$

Lemma 4.18. *Algorithm 4.17 can be implemented with a running time of $2^{O((1/\delta)\log^4(1/\delta))}\log(n)$.*

Proof. In step 1, the algorithm guesses which jobs are processed on machines following a complex configurations. Since each configuration contains at most $O(1/\delta)$ jobs, there are at most $O((1/\delta^4)\log^2(1/\delta))$ jobs assigned to such machines. For each size $\pi_k \in \Pi$, we guess

the number b_k^c of jobs of size π_k assigned to such machines. Hence, we can enumerate all possibilities for jobs assigned to complex machines in time $2^{O((1/\delta)\log^2(1/\varepsilon))}$. Similarly, we can guess the number of machines in step 2 since $m_c \in O((1/\delta^3)\log^2(1/\delta))$. For step 3 we use a simple dynamic program that goes over the machines storing a table $T(\ell, z_1, \ldots, z_d)$ that contains the minimum cost achieved over the first $\ell \leq m^c$ machines with $z_k \leq b_k^c$ jobs of size π_k. The number of entries the table is $O(m^c \prod_{k=1}^{d}(b_k^c + 1))$. Computing $T(\ell, z_1, \ldots, z_d)$ can be done by checking all entries of the type $T(\ell - 1, z'_1, \ldots, z'_d)$ for $z'_k \leq z_k$. Thus, the running time of the dynamic programm is $O(m^c[\prod_{k=1}^{d}(b_k^c + 1)]^2)$. Since $b_k^c \in O((1/\delta^4)\log^2(1/\delta))$ for each k, recalling that $m^c \in O((1/\delta^3)\log^2(1/\delta))$, and that $d = |\Pi| \in O((1/\delta)\log(1/\delta))$, we obtain that step 3 can be implemented with $2^{O((1/\delta)\log^2(1/\delta))}$ running time.

In step 4, our algorithm guesses the support of the solution implied by Theorem 4.16. Let D be the bound implied by the third property of this theorem, so that $|\operatorname{supp}(x) \cap Q_s| \leq D$ and $D \in O((1/\delta)\log^2(1/\delta))$. Also, $|Q_s| \leq 2^{O(\log^2(1/\delta))}$ Then the guessing in step 4 needs to consider the following number of possibilities:

$$\sum_{i=0}^{D} \binom{|Q_s|}{i} \leq (D+1)|Q_s|^D \leq 2^{O((1/\delta)\log^4(1/\delta))}.$$

In step 5, the number of variables of the restricted ILP is $|\bar{Q}_s| = O((1/\delta)\log^2(1/\delta))$. Moreover, using Lemma 4.15 the size of the input is bounded by $O((1/\delta^2)\log^3(1/\delta)\log(n))$. Running Kannan's algorithm [Kan87] to solve the ILP takes time $2^{O((1/\delta)\log^3(1/\delta))}\log(n)$. Hence, the total running time of our algorithm can be bounded by $2^{O((1/\delta)\log^4(1/\delta))}\log(n)$. □

As in [Alo+98], the algorithm above can be easily adapted for objectives (II), (I') and (III') by suitably adapting the ILP. We leave the details to the reader. This suffices to conclude Theorem 4.19

Theorem 4.19. *Consider the scheduling problem on parallel machines with objective functions (I), (II) for f convex (respectively (I') and (II') for f concave). If f satisfies Condition 4.12 for $1/\delta = O(1/\varepsilon)$, then the problem admits an EPTAS with running time $2^{O((1/\varepsilon)\log^4(1/\varepsilon))} + O(n \log n)$.*

4.5 Minimum makespan scheduling on uniform machines

In this section we generalize our result for $P||C_{\max}$ to uniform machines. Consider a set of jobs $\mathcal{J}$ with processing times p_j and a set of m non-identical machines $\mathcal{M}$ where machine $i \in \mathcal{M}$ runs at speed s_i. If job j is executed on machine i the machine needs p_j/s_i time units to complete the job. The problem is to find an assignment $a : \mathcal{J} \to \mathcal{M}$ for the jobs to the machines that minimizes the makespan; $\max_i \sum_{j:a(j)=i} p_j/s_i$. The problem is denoted by $Q||C_{max}$. We suppose that $s_1 \geq s_2 \geq \ldots \geq s_m$. Jansen [Jan10] found an efficient polynomial time approximation scheme (EPTAS) for this scheduling problem which has a running time of $2^{O(1/\varepsilon^2 \log^3(1/\varepsilon))} + \text{poly}(n)$. Here we show how to improve the running time and prove the main result of this section.

Theorem 4.20. *There is an EPTAS (a family of algorithms $\{A_\varepsilon : \varepsilon > 0\}$) which, given an instance I of $Q||C_{max}$ with n jobs and m machines and a positive number $\varepsilon > 0$, produces a schedule of makespan $A_\varepsilon(I) \leq (1+\varepsilon)\textsc{Opt}(I)$. The running time of A_ε is $2^{O(1/\varepsilon \log^4(1/\varepsilon))} + \text{poly}(n)$.*

We follow the approach by Jansen [Jan10], transforming the scheduling problem into a bin packing problem with different bin capacities, round the processing times and bin capacities, divide the bins into at most three groups and generate four different scenarios depending on the input instance.

First, we compute a 2-approximate solution using the algorithm by Gonzales et al. [GIS77] of length $B(I) \leq 2\,\textsc{Opt}(I)$. Suppose that $\varepsilon < 1$; otherwise we can take the 2-approximate solution and are done. Then we choose a value $\delta \in (0, \varepsilon)$ such that $1/\delta$ is integral (the exact value is specified later) and use a binary search within the interval $[B(I)/2, B(I)]$ (that contains $\textsc{Opt}(I)$). We use a standard dual approximation method that for each value T either computes an approximate schedule of length $T(1+a\delta)$ (where a is constant) or shows that there is no schedule of length T. Since $(\delta/2)B(I) \leq \delta\textsc{Opt}(I)$, we can find within $O(\log(1/\delta))$ iterations a value $T \leq \textsc{Opt}(I)(1+\delta)$ with a corresponding schedule of length at most $T(1+a\delta) \leq \textsc{Opt}(I)(1+\varepsilon)$, using $\delta \leq \varepsilon/(a+2)$ and $\varepsilon \leq 1$. Next, the scheduling problem is transformed into a bin packing problem with m bins and capacities $c_i = T \cdot s_i$, the processing times p_j are rounded to the next value $\bar{p}_j$ of the form $\delta(1+\delta)^{k_j}$ with $k_j \in \mathbb{Z}$ and the bin capacities are rounded to the next power of $(1+\delta)$. We call $\mathcal{B}$ the set of bins with rounded capacities.

Lemma 4.21 (Jansen [Jan10])**.** *If there is a feasible packing of n jobs with processing times p_j into m bins with capacities c_i, then there is also a packing of n jobs with rounded processing times $\bar{p}_j = \delta(1+\delta)^{k_j} \leq (1+\delta)p_j$ into m bins with rounded bin capacities $c'_i = (1+\delta)^{\ell_i} \leq c_i(1+\delta)^2$ with $\ell_i \in \mathbb{Z}$.*

If the number m of bins is smaller than $K \in O(1/\delta\log(1/\delta))$, then we can use an approximation scheme by Jansen and Mastrolilli [JM10] to compute an $(1+\varepsilon)$-approximate solution to schedule n jobs on m unrelated machines (an even more general problem) within time $O(n)+2^{O(m\log(m/\varepsilon))} = O(n)+2^{O(1/\delta\log^2(1/\delta))} = O(n)+2^{O(1/\varepsilon\log^2(1/\varepsilon))}$; using that $\delta \in O(\varepsilon)$. Suppose from now on that $K > O(1/\delta\log(1/\delta))$. Then, we divide the bins into at most three different bin groups. The first group $\mathcal{B}_1$ consists of the $K = O(1/\delta\log(1/\delta))$ largest bins. For some $\gamma \in \Theta(\varepsilon^2)$, the next group $\mathcal{B}_2$ consists either of all the remaining bins $\{b_{K+1}, \ldots, b_m\}$ if $c'_m > \gamma c'_K$ (and we have only two bin groups) or $\mathcal{B}_2$ contains the next G largest bins $\{b_{K+1}, \ldots, b_{K+G}\}$ where G is the smallest index such that capacity $c'_{K+G+1} \leq \gamma c'_K$. In the second case, $\mathcal{B}_3 = \{b_{K+G+1}, \ldots, b_m\}$. Let $c_{max}(\mathcal{B})$ and $c_{min}(\mathcal{B})$ be the largest and smallest bin capacity in $\mathcal{B}$. If $c_{max}(\mathcal{B})/c_{min}(\mathcal{B}) \leq C$ for some value C and $\mathcal{B}$ contains only rounded capacities $(1+\delta)^x$ with $x \in \mathbb{Z}$, then the number of different capacities in $\mathcal{B}$ is at most $O(1/\delta\log(C))$.

Lemma 4.22 (Jansen [Jan10])**.** *If there is a solution for the original instance $(\mathcal{J}, \mathcal{M})$ of our scheduling problem with makespan T and corresponding bin sizes, then there is a feasible packing for instance $(\mathcal{J}, \mathcal{B}'_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3)$ or instance $(\mathcal{J}, \mathcal{B}'_1 \cup \mathcal{B}_2)$ with rounded bin capacities $\bar{c}_i \leq c_i(1+\delta)^3$ and rounded processing times $\bar{p}_j \leq (1+\delta)p_j$. Here $\mathcal{B}'_1$ is the subset of $\mathcal{B}_1$ with bins of capacity larger than $\delta/(K-1)c_{max}(\mathcal{B}_1)$ and $\mathcal{B}_2$ has a constant number $O(1/\delta\log(1/\delta))$ of different bin capacities. In addition we have one of the following four scenarios:*

- ***(1)*** *Two bin groups $\mathcal{B}'_1$ and $\mathcal{B}_2$ with a gap $c_{min}(\mathcal{B}'_1)/c_{max}(\mathcal{B}_2) \geq 1/\delta$.*
- ***(2)*** *Two bin groups $\mathcal{B}_1$ and $\mathcal{B}_2$ with a constant number $O(1/\delta\log(1/\delta))$ of different bin capacities in $\mathcal{B}'_1 \cup \mathcal{B}_2$.*

(3) *Three bin groups $\mathcal{B}'_1, \mathcal{B}_2, \mathcal{B}_3$ with a gap $c_{min}(\mathcal{B}'_1)/c_{max}(\mathcal{B}_2) \geq 1/\delta$ and $c_{min}(\mathcal{B}_1)/c_{max}(\mathcal{B}_3) \geq 1/\gamma$.*

(4) *Three bin groups $\mathcal{B}'_1, \mathcal{B}_2, \mathcal{B}_3$ with a constant number $O(1/\delta \log(1/\delta))$ of different bin capacities in $\mathcal{B}'_1 \cup \mathcal{B}_2$ and a gap $c_{min}(\mathcal{B}_1)/c_{max}(\mathcal{B}_3) \geq 1/\gamma$.*

Notice that scenario 4 can be seen as a special case of scenario 3 by using $\mathcal{B}_2^{new} = \mathcal{B}_1 \cup \mathcal{B}_2$, $\mathcal{B}_1^{new} = \emptyset$, and $\mathcal{B}_3^{new} = \mathcal{B}_3$. The same modification works to show that scenario 2 is a special case of scenario 1. Finally, scenario 1 can be interpreted a special case of scenario 3, using $\mathcal{B}_3 = \emptyset$. Therefore, it is sufficient to improve the running time for scenario 3.

Scenario 3 will be solved using a mix of dynamic programming and mixed integer linear programming (MILP) techniques. In this approach we use only the larger bins in $\mathcal{B}'_1$ of $\mathcal{B}_1$ to execute jobs, but in the rounding step for scenario 3 afterwards we may also use the smaller bins in $\mathcal{B}_1$. Notice that a packing into a bin b_i with capacity $\bar{c}_i \leq c_i(1+\delta)^3$ corresponds to a schedule on machine i with total processing time at most $\bar{c}_i/s_i \leq T(1+\delta)^3$. For $T \leq \textsc{Opt}(I)(1+\delta)$ this gives us a schedule of length at most $\textsc{Opt}(I)(1+\delta)^4$. If there a feasible schedule with makespan T, then the total processing time of the instance is $\sum_{j\in\mathcal{J}} p_j \leq \sum_{i=1}^{m} \bar{c}_i$. If this inequality does not hold, then we discard the choice with makespan T. Otherwise, we can eliminate the set $\mathcal{J}_{\text{tiny}}$ of tiny jobs with processing time $\leq \delta\bar{c}_m$ and pack them greedily at the end of the algorithm into the enlarged bins of size $\bar{c}_i(1+\delta)$. Hence, in what follows we assume that $\mathcal{J}_{\text{tiny}}$ is empty.

4.5.1 Solution for the instance $(\mathcal{J}, \mathcal{B}'_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3)$

In this subsection we consider scenarios 3 above with three bin groups. First, we preassign all huge jobs with processing time $> \delta\bar{c}_{K'}$ into the first K' machines. Since $\bar{c}_{K+1} = c_{max}(\mathcal{B}_2) \leq \delta c_{min}(\mathcal{B}'_1) = \delta\bar{c}_{K'}$, the huge jobs fit only on the first K' bins. The number of huge jobs can be bounded by $O(Kc_{max}(\mathcal{B}_1)/(\delta c_{K'})) = O(1/\delta^4 \log^2(1/\delta))$. If there are more huge jobs in the instance, then there is no packing into $\mathcal{B}'_1 \cup \mathcal{B}_2$ and we are done. Furthermore, the number of machines $K' \in O(1/\delta \log(1/\delta))$ is constant. Again, we can use the approximation scheme by Jansen and Mastrolilli that computes an $(1+\delta)$-approximate schedule for N jobs on M machines which runs in $O(N) + 2^{O(M \log(M/\delta))}$. For $M \in O(1/\delta \log(1/\delta))$ and $N \in O(1/\delta^4 \log^2(1/\delta))$ this gives a running time $O(1/\delta^4 \log^2(1/\delta)) + 2^{O(1/\delta \log^2(1/\delta))} = 2^{O(1/\delta \log^2(1/\delta))}$ to obtain a feasible packing with bin sizes $\bar{c}_i(1+\delta)$ or schedule of length $\leq T(1+\delta)^4$, if one exists. If there is no feasible packing for the huge jobs, then there is no schedule with makespan T and we have to increase T in the binary search. In the other case we set up a MILP.

After the assignment of the huge jobs, we have a free area S_0 in $\mathcal{B}_1$ for the remaining jobs with processing time $\bar{p}_j \leq \delta\bar{c}_{K'}$. The different bin capacities in $\mathcal{B}_2$ and $\mathcal{B}_3$ are denoted by $\bar{c}(1) > \ldots > \bar{c}(L)$ and $\bar{c}(L+1) > \ldots > \bar{c}(L+N)$, respectively. Let m_ℓ be the number of bins of size $\bar{c}(\ell)$ for $\ell = 1, \ldots, L+N$. The m_ℓ machines of the same speed form a block B_ℓ of bins with the same capacity $\bar{c}(\ell)$. In addition, we have $n_1, \ldots, n_P$ jobs of size $\delta(1+\delta)^{k_j}$ and suppose that the first $P' \leq P$ job sizes are larger than $\bar{c}_{K+1} = \bar{c}(1)$.

In the MILP we use $C_1^{(\ell)}, \ldots, C_{h_\ell}^{(\ell)}$ as configurations or multisets with numbers $\delta(1+\delta)^{k_j} \in [\delta\bar{c}(\ell), \bar{c}(\ell)]$ (these are large processing times corresponding to block B_ℓ), where the total sum $size(C_i^{(\ell)}) = \sum_j a(k_j, C_i^{(\ell)})\delta(1+\delta)^{k_j}$ is bounded by $\bar{c}(\ell)$. Here $a(k_j, C_i^{(\ell)})$ is the number of occurrences of number $\delta(1+\delta)^{k_j}$ in configuration $C_i^{(\ell)}$. In the MILP below, we use integral and fractional variables $x_i^{(\ell)}$ to indicate number of machines that are

scheduled according to configuration $C_i^{(\ell)}$. In addition, we use fractional variables $y_{j,\ell}$ to indicate the number of jobs of size $\delta(1+\delta)^{k_j}$ placed as small ones in block B_ℓ; i.e. $\delta(1+\delta)^{k_j} < \delta\bar{c}(\ell)$. For each job size $\delta(1+\delta)^{k_j} \leq \bar{c}(1)$, let a_j be the smallest index in $\{1,\ldots,L+N\}$ such that $\delta(1+\delta)^{k_j} \geq \delta\bar{c}(a_j)$. If there is no such index, we have a tiny processing time $\delta(1+\delta)^{k_j} < \delta\bar{c}(L+N)$. Notice that the first P' job sizes are within $(\bar{c}(1), \delta c_{K'}]$. These jobs do not fit into $\mathcal{B}_2 \cup \mathcal{B}_3$. Therefore, for these job sizes we use only one variable $y_{j,0} = n_j$ and set $a_j = 0$.

$$
\begin{array}{lll}
\sum_i x_i^{(\ell)} & \leq m_\ell & \text{for} \quad \ell = 1,\ldots,L+N, \\
\sum_{\ell,i} a(k_j, C_i^{(\ell)}) x_i^{(\ell)} + \sum_{\ell=0}^{a_j-1} y_{j,\ell} & = n_j & \text{for} \quad j = P'+1,\ldots,P, \\
\sum_i size(C_i^{(\ell)}) x_i^{(\ell)} + \sum_{j:\ell<a_j} y_{j,\ell}\delta(1+\delta)^{k_j} & \leq m_\ell\bar{c}(\ell) & \text{for} \quad \ell = 1,\ldots,L+N, \\
\sum_{j=1}^{P} y_{j,0}\delta(1+\delta)^{k_j} & \leq S_0, &
\end{array}
$$

$$
\begin{array}{ll}
x_i^{(\ell)} \text{ integral} \geq 0 & \text{for } \ell = 1,\ldots,L \text{ and } i = 1,\ldots,h_\ell, \\
x_i^{(\ell)} \geq 0 & \text{for } \ell = L+1,\ldots,L+N \text{ and } i = 1,\ldots,h_\ell \\
y_{j,0} = n_j & \text{for } j = 1,\ldots,P', \\
y_{j,\ell} \text{ integral} \geq 0 & \text{for } j = P'+1,\ldots,P \text{ and } \ell = 0,\ldots,a_j-1.
\end{array}
$$

In the MILP above, we use integral variables for configurations in the blocks of group $\mathcal{B}_2$ and fractional variables for the configurations in blocks of $\mathcal{B}_3$. Each feasible packing for the jobs into the bins corresponds to a feasible solution of the MILP. The total number of variables is $O(n^2) + O(n)2^{O(1/\delta\log(1/\delta))}$, the number of integral variables is at most $2^{O(1/\delta\log(1/\delta))}$, and the number of constraints (not counting the non-negativity constraints) is at most $O(n)$. The previous approach to solve the scheduling problem and the underlying MILP had a running time of $2^{O(1/\delta^2\log^3(1/\delta))} + \text{poly}(n)$. In order to use an approach similar to the scheduling on identical machines, each large size $\delta(1+\delta)^{k_j} \in C_i^{(\ell)}$ is rounded up to the next multiple of $\delta^2\bar{c}(\ell)$. This enlarges the size of each configuration $C_i^{(\ell)}$ from $size(C_i^{(\ell)})$ to at most $size(C_i^{(\ell)}) + \delta\bar{c}(\ell)$ and the corresponding bin size from $\bar{c}(\ell)$ to $(1+\delta)\bar{c}(\ell)$.

Let $\bar{C}_1^{(\ell)},\ldots,\bar{C}_{\bar{h}_\ell}^{(\ell)}$ be the configurations of size at most $(1+\delta)\bar{c}(\ell)$ with the rounded-up numbers $q(k_j,\ell)\delta^2\bar{c}(\ell)$ with $q(k_j,\ell) \in \mathbb{Z}^+$ and multiplicities $a(k_j, \bar{C}_i^{(\ell)})$. This rounding implies also that the rounded size $size(\bar{C}_i^{(\ell)})$ of a configuration is a multiple of $\delta^2\bar{c}(\ell)$. Each new rounded configuration $\bar{C}_i^{(\ell)}$ (with rounded-up numbers $q(k_j,\ell)\delta^2\bar{c}(\ell)$ and multiplicities $a(k_j, \bar{C}_i^{(\ell)})$) corresponds to an integral point inside the knapsack polytope $\mathcal{P}_\ell = \{C = (a(k_j, C)) : q \cdot C \leq 1/\delta^2 + 1/\delta\}$ such that $\sum_j q(k_j,\ell)a(k_j,\bar{C}_i^{(\ell)})\delta^2\bar{c}(\ell) = size(\bar{C}_i^{(\ell)}) \leq (1+\delta)\bar{c}(\ell)$ or, equivalently, $\sum_j q(k_j,\ell)a(k_j,\bar{C}_i^{(\ell)}) \leq 1/\delta^2 + 1/\delta \leq 2/\delta^2$. We consider now a modified MILP with configurations $\bar{C}_i^{(\ell)}$ and coefficients $a(k_j, \bar{C}_i^{(\ell)})$. Note that the total area of all configurations in B_ℓ can be bounded by $\sum_i size(\bar{C}_i^{(\ell)})x_i^{(\ell)} \leq \sum_i size(C_i^{(\ell)})x_i^{(\ell)} + \delta\bar{c}(\ell)\sum_i x_i^{(\ell)}$. This, together with the small jobs gives $\sum_i size(\bar{C}_i^{(\ell)})x_i^{(\ell)} + \sum_j \delta(1+\delta)^{k_j}y_{j,\ell} \leq \sum_i size(C_i^{(\ell)})x_i^{(\ell)} + \delta m_\ell\bar{c}(\ell) + \sum_j \delta(1+\delta)^{k_j}y_{j,\ell} \leq m_\ell\bar{c}(\ell)(1+\delta)$; i.e. the total area is increased by at most a multiplicative factor of $(1+\delta)$. Since the total area of all jobs within one block is increased by this rounding, we use the following new constraints in the modified MILP:

$$
\sum_i size(\bar{C}_i^{(\ell)})x_i^{(\ell)} + \sum_j y_{j,\ell}\delta(1+\delta)^{k_j} \leq m_\ell\bar{c}(\ell)(1+\delta) \qquad \text{for } \ell = 1,\ldots,L.
$$

Next, we divide the coefficients in the L area constraints above by $\delta^2\bar{c}(\ell)$. Then the coefficients of the $x_i^{(\ell)}$ variables are now $size(\bar{C}_i^{(\ell)})/(\delta^2\bar{c}(\ell)) = a_{i,\ell}\delta^2\bar{c}(\ell)/(\delta^2\bar{c}(\ell)) = a_{i,\ell} \in \{1/\delta, \ldots, 1/\delta^2 + 1/\delta\}$. Using the assumption that $1/\delta$ is integral, all coefficients of the variables are integral and bounded by $2/\delta^2$. Notice that increasing the capacities of all bins and dividing all coefficients as above, implies also a feasible solution of the modified MILP. Let us study a feasible solution of the modified MILP. To reduce the number of integral configuration variables in the MILP, we consider the following ILP that uses only the integral $x_i^{(\ell)}$ variables within bin group $\mathcal{B}_2$:

$$\sum_i x_i^{(\ell)} = \bar{m}_\ell \qquad \text{for } \ell = 1, \ldots, L, \tag{4.10}$$

$$\sum_{\ell,i} a(k_j, \bar{C}_i^{(\ell)}) x_i^{(\ell)} = \bar{n}_j \qquad \text{for } j \in P(\mathcal{B}_2), \tag{4.11}$$

$$\sum_i \frac{size(\bar{C}_i^{(\ell)})}{\delta^2\bar{c}(\ell)} x_i^{(\ell)} = Area(\ell, large) \qquad \text{for } \ell = 1, \ldots, L, \tag{4.12}$$

$$x_i^{(\ell)} \text{ integral} \geq 0 \qquad \text{for } i = 1, \ldots, \bar{h}_\ell, \ell = 1, \ldots, L. \tag{4.13}$$

where the values $\bar{m}_\ell$, $\bar{n}_j$, and $Area(\ell, large)$ are given by a feasible solution of the modified MILP. Here $P(\mathcal{B}_2)$ is the set of all indices of large job sizes corresponding to blocks $B_\ell \in \mathcal{B}_2$; i.e. $P(\mathcal{B}_2) = \{j : \delta(1+\delta)^{k_j} \in (\delta\bar{c}(L), \bar{c}(1)]\}$. The cardinality of $P(\mathcal{B}_2)$ and the value L can be bounded by $O(1/\delta \log(1/\delta))$. All the coefficients above of the variables are bounded by $O(1/\delta^2)$.

The support of a configuration $\bar{C}_i^{(\ell)}$ is the number of values $a(k_j, \bar{C}_i^{(\ell)}) > 0$; i.e. $\text{supp}(\bar{C}_i^{(\ell)}) = |\{j : a(k_j, \bar{C}_i^{(\ell)}) > 0\}|$. In our case $\text{supp}(\bar{C}_i^{(\ell)})) \leq O(1/\delta \log(1/\delta))$. A configuration $\bar{C}_i^{(\ell)}$ is called simple, if $|\text{supp}(\bar{C}_i^{(\ell)})| \leq \log(1/\delta^2 + 1/\delta + 1)$ Otherwise, we call a configuration $\bar{C}_i^{(\ell)}$ complex. Using the result by Eisenbrand and Shmonin, we can find a feasible solution of the ILP above (if there is a feasible solution of the modified MILP) with at most $O(1/\delta \log^2(1/\delta))$ many variables $x_i^{(\ell)} > 0$; i.e. $|\text{supp}(x)| \leq O(1/\delta \log^2(1/\delta))$ where $x = (x_i^{(\ell)})$. We can generalize our result in Theorem 4.1 to our ILP above.

Lemma 4.23. *Assume that the ILP defined by* (4.10)-(4.13) *is feasible and let S denote the set of all simple configurations. Then there exists a feasible solution x' such that:*

(1) *If $x'^{(\ell)}_i > 1$ then the configuration $\bar{C}_i^{(\ell)}$ is simple.*

(2) *The support of x' satisfies $|\text{supp}(x') \cap S| \in O(1/\delta \log^2(1/\delta))$.*

(3) *The support of x' satisfies $|\text{supp}(x') \setminus S| \in O(1/\delta^2 \log^3(1/\delta))$.*

Proof. As stated above, the set of configurations $\bar{C}_1^{(\ell)}, \ldots, \bar{C}_{\bar{h}_\ell}^{(\ell)}$ equals the set of integral points Q_ℓ inside the knapsack polytope $\mathcal{P}_\ell = \{C = (q(k_j, C)) : q \cdot C \leq 1/\delta^2 + 1/\delta\}$ Let $\bar{x} = (\bar{x}^{(\ell)})_{\ell=1}^L$ be a solution to (4.10)-(4.13) where $\bar{x}^{(\ell)}$ corresponds to the variables defining the solution for block B_ℓ. We consider a family of ILPs defined for each $\ell = 1, \ldots, L$.

$$\begin{aligned} [\text{conf-IP}]_\ell \quad & \sum_{c \in Q_\ell} c \cdot x_c = \sum_{c \in Q_\ell} c \cdot \bar{x}_c^{(\ell)}, \\ & \sum_{c \in Q_\ell} x_c = \bar{m}_\ell, \\ & x_c \in \mathbb{Z}_{\geq 0} \qquad \text{for all } c \in Q_\ell. \end{aligned}$$

Using Theorem 4.1 for each [conf-IP]$_\ell$, we obtain new solution $\hat{x}^{(\ell)}$, where each complex configuration is used at most once and $\operatorname{supp}(\hat{x}^{(\ell)}) \in O(1/\delta \log^2(1/\delta))$. Then we define a new solution $\hat{x}$ of ILP (4.10)-(4.13) defined as $(\hat{x}^{(\ell)})_\ell$. In $\hat{x}$ every complex configuration is used at most once and $|\operatorname{supp}(\hat{x})| \leq L \cdot 2(\bar{d}+1)\log(4(\bar{d}+1)\bar{T}) \in O(1/\delta^2 \log^3(1/\delta))$, where $\bar{d} \leq |P(\mathcal{B}_2)| \in O(1/\delta \log 1/\delta)$ and $\bar{T} \in O(1/\delta^2)$. Note that Equation (4.12) of the above ILP holds for the new solution $\hat{x}$ as the set of jobs covered inside a block does not change and hence

$$\sum_i \frac{size(\bar{C}_i^{(\ell)})}{\delta^2 \bar{c}(\ell)} \hat{x}_i^{(\ell)} = Area(\ell, large) = \sum_i \frac{size(\bar{C}_i^{(\ell)})}{\delta^2 \bar{c}(\ell)} \bar{x}_i^{(\ell)}. \qquad \square$$

Finally, consider the ILP (4.10)-(4.13) and fix each variable x_i^ℓ, for $\bar{C}_i^{(\ell)}$ a complex configuration, to the value $\hat{x}_i^\ell$ (and thus the resulting ILP has variables only for simple configurations). Now we can apply the result of Eisenbrand and Shmonin [ES06] to this ILP. This ensures that any ILP of the form $\{z \in \mathbb{Z}_{\geq 0} : Az = h\}$ admits a solution with support of size $O(N(\log(N) + \Delta))$, where N is the number of rows of A and Δ is the largest encoding size of an entry of A. Recalling that $\frac{size(\bar{C}_i^{(\ell)})}{\delta^2 \bar{c}(\ell)} \in O(1/\delta^2)$, we can apply this result to our case, which yields a solution whose support contains at most $O(1/\delta \log^2(1/\delta))$ simple configurations. Hence, we obtain a solution satisfying all properties of the statement of the theorem.

Algorithm 4.24.

1. *For each job size, guess the number of jobs $v_j \leq \bar{n}_j$ covered by complex configurations.*
2. *For each bin size, guess the number of machines $w_j \leq \bar{m}_j$ used to schedule the set of jobs covered by complex configurations.*
3. *For each block ℓ in $\mathcal{B}_2$, guess the support of simple configurations $\bar{Q}_s^{(\ell)} \subseteq Q_s^{(\ell)}$ used by a thin solution, with $\sum_{\ell=1}^{L} |\bar{Q}_s^{(\ell)}| \leq 4(d+1)\log(4(d+1)\bar{T}) \in O((1/\varepsilon)\log^2(1/\varepsilon))$.*
4. *Solve the reduced modified MILP, where the integral variables x_i^ℓ are restricted to simple configurations.*

Lemma 4.25. *Algorithm 4.24 can be implemented with a running time of $2^{O((1/\varepsilon)\log^4(1/\varepsilon))} poly(n)$.*

Proof. As in the case of identical machines, our algorithm guesses in step 1 the complex configurations and the corresponding jobs. Since the number M of complex configurations within $\mathcal{B}_2$ ist at most $O(1/\delta^2 \log^3(1/\delta))$ and there are at most $O(1/\delta)$ many large job per configuration, the total number N of jobs within the complex configurations is at most $O(1/\delta^3 \log^3(1/\delta))$.

To obtain a schedule for the guessed jobs, notice that the number of large job sizes $|P(\mathcal{B}_2)| \leq O(1/\delta \log(1/\delta))$. We guess now a vector $v = (v_j)$ with possible job sizes that are covered by the complex configurations. The total number of these vectors is $(N+1)^{|P(\mathcal{B}_2)|} \leq (1/\delta^3 \log^3(1/\delta))^{O(1/\delta \log(1/\delta))} = 2^{O(1/\delta \log^2(1/\delta))}$. In addition, we guess a vector $w = (w_\ell)$ with the numbers w_ℓ of complex configurations in the block groups B_ℓ. The number of choices here is at most $(M+1)^L \leq (1/\delta^2 \log^3(1/\delta))^{O(1/\delta \log(1/\delta))} = 2^{O(1/\delta \log^2(1/\delta))}$. For each guess v, w we run a dynamic program to test whether the number of job sizes, stored in v, fit on the corresponding machines in the blocks B_ℓ, given by vector w. To do this, we run over the machines and store after ℓ machines, for $\ell = 1, \ldots, M$, the set of all

feasible vectors with job sizes that can be packed into the first ℓ machines. This dynamic program runs in time $M2^{O(1/\delta \log^2(1/\delta))} = 2^{O(1/\delta \log^2(1/\delta))}$. For each feasible choice of v, w we compute the reduced MILP by $\hat{m}_\ell = m_\ell - w_\ell$ and $\hat{n}_j = n_j - v_j$ and guess the support of a feasible solution x in the MILP; i.e. the simple configurations in $\mathcal{B}_2$ with value $x_i^{(\ell)} > 0$. The total number of simple configurations $Q_s^{(\ell)}$ in one bin block can be bounded, using observation 7, by $2^{O(\log^2(1/\delta))}$. Therefore, the total number of simple configurations in $\mathcal{B}_2$ is $\sum_{\ell=1}^{L} |Q_s^{(\ell)}| \leq L \cdot 2^{O(\log^2(1/\delta))} = 2^{O(\log^2(1/\delta))}$. This implies that the number of choices for the support of x is at most

$$\binom{\sum_\ell |Q_s^{(\ell)}|}{O(1/\delta \log^2(1/\delta))} = \binom{2^{O(\log^2(1/\delta))}}{O(1/\delta \log^2(1/\delta))} = 2^{O(1/\delta \log^4(1/\delta))}.$$

For each choice we solve a reduced MILP with $d = O(1/\delta \log^2(1/\delta))$ integral variables (step 4). The total size s of the MILP can be bounded by $s \leq poly(n, 1/\delta) + n \log(n) 2^{O(1/\delta \log(1/\delta))}$. Using the algorithm by Kannan with runnning time $d^{O(d)} poly(s)$ for an MILP with d variables and size s, we obtain a running time to solve one MILP in time $2^{O(1/\delta \log^3(1/\delta))} poly(n)$; using $poly(s) \leq poly(n) 2^{O(1/\delta \log(1/\delta))}$. Running over all vectors v, w and all guesses for the simple configurations, we obtain a running time of $2^{O(1/\delta \log^4(1/\delta))} + poly(n)$. □

The rounding of the fractional variables in the MILP solutions and the packing of the items accordingly works as in [Jan10] and can be done in time $2^{O(1/\delta \log(1/\delta))} poly(n)$. Therefore, the overall running time of the entire algorithm can be bounded by $2^{O(1/\delta \log^4(1/\delta))} + poly(n) = 2^{O(1/\varepsilon \log^4(1/\varepsilon))} + poly(n)$; using that $\delta \in O(\varepsilon)$.

In order to calculate the length of the computed schedule and to specify δ, we use the following result:

Lemma 4.26. *[Jan10] If there is a feasible solution of an MILP instance with bin capacities $\bar{c}(\ell)$ for blocks $B_\ell \in \mathcal{B}_2 \cup \mathcal{B}_3$ and capacities $\bar{c}_i$ for the K largest bins in $\mathcal{B}_1$, then the entire job set $\mathcal{J}$ can be packed into bins with capacities $\bar{c}(\ell)(1+2\delta)^2$ for blocks $B_\ell \in \mathcal{B}_2 \cup \mathcal{B}_3$ and enlarged capacities $\bar{c}_i(1+3\delta)^2$ for the first K bins.*

Note that the result above is constructive, too. This means that there is also an algorithm that computes a corresponding packing [Jan10]. Using $\bar{c}_i \leq c_i(1+\delta)^3$ and $T \leq (1+\delta)OPT$ and the lemma above, we can bound the schedule length. If there is a schedule with length at most T and with corresponding bin sizes $c_i = Ts_i$, then the lemma above implies a packing into bins of size $c_i(1+3\delta)^3(1+3\delta)^2$ and a corresponding schedule length $\leq T(1+\delta)^3(1+3\delta)^2 \leq OPT(1+\delta)^4(1+3\delta)^2 \leq OPT(1+16\delta) \leq OPT(1+\varepsilon)$ for $\delta \leq \varepsilon/16$ and $\varepsilon \leq 1$. Using $\delta = \frac{1}{\lceil 16/\varepsilon \rceil}$, we obtain $\delta \leq \varepsilon/16$, $\delta \geq \varepsilon/17$, and that $1/\delta = \lceil 16/\varepsilon \rceil$ is integral. This concludes the proof for Theorem 4.20.

5 About the Structure of the Integer Cone and its Application to Bin Packing

5.1 Introduction

Given the polytope $\mathcal{P} = \{x \in \mathbb{R}^d \mid Ax \leq c\}$ for some matrix $A \in \mathbb{Z}^{m \times d}$ and a vector $c \in \mathbb{Z}^d$. We consider the integer cone

$$int.cone(\mathcal{P} \cap \mathbb{Z}^d) = \{ \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p \mid \lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d} \}$$

of integral points inside the polytope $\mathcal{P}$. Let $\mathcal{P}_I = Conv(\mathcal{P} \cap \mathbb{Z}^d)$ be the convex hull of all integer points inside $\mathcal{P}$, where for given set $X \subset \mathbb{R}^d$, the convex hull of X is defined by $Conv(X) = \{\sum_{p \in X} x_p p \mid x \in [0,1]^X, \|x\|_1 = 1\}$. Let V_I be the vertices of the integer polytope $\mathcal{P}_I$ i.e. $\mathcal{P}_I = Conv(V_I)$. In case of the (fractional) cone $Cone(\mathcal{P} \cap \mathbb{Z}^d) = \{\sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p \mid \lambda \in \mathbb{R}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}\}$, we know by Caratheodory's Theorem (see e.g. [Sch86]) that each $\gamma \in \mathcal{P}_I$ can be written as a convex combination of at most $d+1$ points in V_I and hence $Cone(\mathcal{P} \cap \mathbb{Z}^d) = Cone(V_I)$.

In this chapter we investigate the structure of the integer cone $int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ versus $int.cone(V_I)$. Therefore, we define the *vertex distance* of a point $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ which describes how many extra points from $(\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$ are needed to represent b. The vertex distance is defined by

$$Dist(b) = \min\{\|\gamma\|_1 \mid \gamma \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}, \lambda \in \mathbb{Z}_{\geq 0}^{V_I} \text{ such that } b = \sum_{v \in V_I} \lambda_v v + \sum_{p \in \mathcal{P}_I} \gamma_p p\}$$

In this chapter we show that for every point $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$, the vertex distance $Dist(b)$ is bounded by $2^{2^{O(d)}}$. Hence, every b can be written by $b = \sum_{v \in V_I} \lambda_v v + \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \gamma_p p$ for some $\lambda \in \mathbb{Z}_{\geq 0}^{V_I}$ and some $\gamma \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$, where $\|\gamma\|_1 \leq 2^{2^{O(d)}}$.

A related result concerning the structure of the integer cone was given by Eisenbrand and Shmonin [ES06]. They proved that every $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ can be written by a vector $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$ with $b = \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p$ such that λ has a bounded support (=number of non-zero components). Therefore, for a given set M and given vector $\lambda \in \mathbb{R}_{\geq 0}^M$, let $supp(\lambda)$ be the set of non-zero components of λ, i.e. $supp(\lambda) = \{s \in M \mid x_s \neq 0\}$.

Theorem 5.1 (Eisenbrand, Shmonin [ES06])**.** *Given polytope $\mathcal{P} \subset \mathbb{R}^d$. For any integral point $b \in int.cone(P \cap \mathbb{Z}^d)$, there exists an integral vector $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$ such that $b = \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p$ and $|supp(\lambda)| \leq 2^d$.*

Let (s, b) be an instance of the bin packing problem with item sizes $s_1, \ldots, s_d \in (0, 1]$ and multiplicities $b \in \mathbb{Z}_{\geq 0}^d$ of the respective item sizes. The objective of the bin packing problem is to pack all items b into as few unit sized bins as possible. When we choose

$\mathcal{P}$ to be the knapsack polytope, i.e. $\mathcal{P} = \{x \in \mathbb{Z}^d_{\geq 0} \mid s^T x \leq 1\}$, then a vector $\lambda \in \mathbb{Z}^{\mathcal{P}_I}_{\geq 0}$ of $int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ yields a packing for the bin packing problem. A long standing open question was, if the bin packing problem can be solved in polynomial time when the number of different item sizes d is constant. This problem was recently solved by Goemans and Rothvoß [GR14] using similar structural properties of the integer cone. They proved the existence of a distinguished set $X \subset \mathcal{P}$ of bounded size such that for every vector $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ there exists an integral vector $\lambda \in \mathbb{Z}^{\mathcal{P} \cap \mathbb{Z}^d}_{\geq 0}$ where most of the weight lies in X. More precisely, they proved the following structure theorem:

Theorem 5.2 (Goemans, Rothvoß [GR14])**.** *Let $\mathcal{P} = \{x \in \mathbb{R}^d_{\geq 0} \mid Ax \leq c\}$ be a polytope with $A \in \mathbb{Z}^{m \times d}, c \in \mathbb{Z}^d$ such that all coefficients are bounded by Δ in absolute value. Then there exists a set $X \subseteq \mathcal{P} \cap \mathbb{Z}^d$ such that for any point $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$, there exists an integral vector $\lambda \in \mathbb{Z}^{\mathcal{P} \cap \mathbb{Z}^d}_{\geq 0}$ such that $b = \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p$ and*

1. $\lambda_p \leq 1 \qquad \forall p \in (\mathcal{P} \cap \mathbb{Z}^d) \setminus X$
2. $|supp(\lambda) \cap X| \leq 2^{2d}$
3. $|supp(\lambda) \setminus X| \leq 2^{2d}$

The set X is constructed in [GR14] by covering $\mathcal{P}$ by a set of integral parallelepipedes. The set X consists of the vertices of the integral parallelepiped and can be computed in a preprocessing step. Note that by the construction of Goemans and Rothvoß, we have that $X \subset V_I$ as the set of vertices of some inner centrally symmetric polytopes is computed.

5.1.1 Our results:

At first, we study the special case when $\mathcal{P}$ is given by the convex hull of integral points $B_0, B_1, \ldots, B_d \in \mathbb{Z}^d$ i.e. $\mathcal{P}$ is the simplex $S = Conv(B_0, B_1, \ldots, B_d)$. This is for example the case in the knapsack polytope when all items sizes are of the form $s_i = 1/a_i$ for some $a_i \in \mathbb{Z}_{\geq 1}$. In this case, all vertices of the knapsack polytope are of the form $B_0 = (0, \ldots, 0)^T$ and $B_i = (0, \ldots, 0, a_i, 0, \ldots, 0)^T$ for $1 \leq i \leq d$ and therefore integral. We prove the following theorem:

Theorem 5.3. *Let S be the simplex defined by $S = Conv(B_0, B_1, \ldots, B_d)$ for $B_i \in \mathbb{Z}^d$ and let B be the set of vertices $B = \{B_0, B_1, \ldots, B_d\}$. For any vector $b \in int.cone(S \cap \mathbb{Z}^d)$, there exists an integral vector $\lambda \in \mathbb{Z}^{S \cap \mathbb{Z}^d}_{\geq 0}$ with $b = \sum_{s \in S \cap \mathbb{Z}^d} \lambda_s s$ and*

1. $\lambda_s \leq 2^{2^{O(d)}} \quad \forall s \in (S \cap \mathbb{Z}^d) \setminus B$
2. $|supp(\lambda) \setminus B)| \leq 2^d$

This theorem shows that in the case that the integer polytope $\mathcal{P}_I$ is a simplex, the vertex distance $Dist(b)$ can be bounded by a term $2^{2^{O(d)}}$ for any $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$. In Section 5.3, we complement this result by giving a matching lower bound for $Dist(b)$. We prove that the double exponential bound for $Dist(b)$ is tight, even in the special case of bin packing, where the simplex S is a specific knapsack polytope. The lower bound is based on the sylvester sequence S_i which is inductively defined by $S_1 = 2$ and $S_i = (\prod_{j=1}^{i} S_i) + 1$ [GKP94].

Theorem 5.4. *There exists a bin packing instance with sizes* $\frac{1}{a_1}, \ldots, \frac{1}{a_d}$ *for* $a_i \in \mathbb{Z}_{\geq 1}$ *and multiplicities* $b \in \mathbb{Z}^d_{\geq 0}$ *corresponding to a point* $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$*, where* $\mathcal{P}$ *is the knapsack polytope such that*

$$Dist(b) \geq S_d - 2 = 2^{2^{\Omega(d)}}$$

Furthermore, in the end of Section 4, we discuss the difficulty of finding instances with large vertex distance and we show a connection to the modified roundup property (see [ST97]).

As a direct consequence of our main Theorem 5.3, we obtain a structure theorem that is similar to the one given by Goemans and Rothoß[GR14] but uses a different set $X \subset \mathcal{P}$ of distinguished points. Instead of the set of vertices of integral parallelepipedes, our theorem uses the set of vertices V_I of the integer polytope.

Theorem 5.5. *Let* $P = \{x \in \mathbb{R}^d_{\geq 0} \mid Ax \leq c\}$ *be a polytope with* $A \in \mathbb{Z}^{m \times d}, c \in \mathbb{Z}^d_{\geq 0}$ *and let* $V_I \subseteq \mathcal{P} \cap \mathbb{Z}^d$ *be the set of vertices of the integer polytope* P_I *with* $Conv(V_I) \cap \mathbb{Z}^d = \mathcal{P} \cap \mathbb{Z}^d$*. Then for any vector* $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$*, there exists an integral vector* $\lambda \in \mathbb{Z}^{\mathcal{P} \cap \mathbb{Z}^d}_{\geq 0}$ *such that* $b = \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p$ *and*

1. $\lambda_p \leq 2^{2^{O(d)}} \quad \forall p \in (\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$
2. $|supp(\lambda) \cap V_I| \leq d \cdot 2^d$
3. $|supp(\lambda) \setminus V_I| \leq 2^{2d}$

This theorem finally shows that for arbitrary polytopes $\mathcal{P}$ and any $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$, the vertex distance $Dist(b)$ is bounded by $2^{2^{O(d)}}$ and hence independent of the number of inequalities m and the largest entry Δ in the description of $\mathcal{P}$.

Recall that a parameterized problem with parameter p and input I is called fixed parameter tractable (fpt) if there exists an algorithm with running time $O(f(p) \cdot enc(I)^{O(1)})$ for some computable function f of p which is independent of I and $enc(I)$ is the encoding length of instance I. We refer to the book of Downey and Fellows [DF99] for more details on parameterized complexity. As a consequence of our structure theorem, we present in Section 2 an algorithm for the bin packing problem with a running time of $|V_I|^{2^{O(d)}} \cdot \log(\Delta)^{O(1)}$, where Δ is the maximum over all multiplicities b and denominators in s. Since $|V_I| \geq d + 1$ this is an fpt-algorithm parameterized by the number of vertices of the integer knapsack polytope V_I.

Theorem 5.6. *The bin packing problem can be solved in fpt-time parameterized by the number of vertices* V_I *of the integer knapsack polytope.*

This theorem shows that the bin packing problem can be solved efficiently when the underlying knapsack polytope has an easy structure i.e. has not too many vertices. However, since the total number of vertices is bounded by $O(\log \Delta)^d$ [HL83] the algorithm has a worst case running time of $(\log \Delta)^{2^{O(d)}}$, which is identical to the running time of the algorithm by Goemans and Rothvoß [GR14].

5.1.2 Related results

The bin packing problem is one of the most fundamental combinatorial problems in computer science. It has been very well studied in the literature, mostly in the context of approximation. A major contribution was given by Karmarkar and Karp [KK82]. They presented

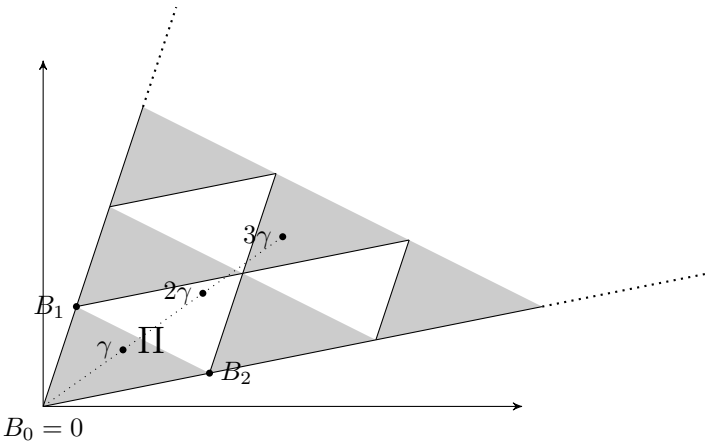


Figure 5.1: Partitioning $Cone(B)$

a polynomial time approximation algorithm with a guarantee of $OPT + O(\log^2(OPT))$. Very recently, this famous result by Karmarkar and Karp was improved by Rothvoß [Rot13] who presented an algorithm with guarantee $OPT + O(\log OPT \log\log(OPT))$ and later by Rothvoßand Hoberg [HR15] who improved the guarantee further to $OPT + O(\log(OPT))$. Concerning the bin packing problem when the number of different item sizes d is constant, Jansen and Solis-Oba [JS11] presented an approximation algorithm with a guarantee of $OPT + 1$. Their algorithm has a running time of $2^{2^{O(d)}} \cdot enc(I)^{O(1)}$ and therefore is fpt in the number of different item sizes d. Finally, as mentioned above, Goemans and Rothvoß [GR14] presented their polynomial time algorithm for the bin packing problem with running time $(\log \Delta)^{2^{O(d)}}$.

In a very recent work, Onn [Onn15] discussed the problem of finding a vector $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$ with $b = \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p$ for given $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$. He presented an algorithm for the case that the polytope $\mathcal{P} = \{x \in \mathbb{R}^d \mid Ax \leq c\}$ has a specific shape. In the case that the matrix A is totally unimodular he gave a polynomial time algorithm even in the case that the dimension d is variable.

5.2 Proof of the main theorem

Given simplex $S = Conv(B_0, B_1, \ldots, B_d)$ for $B_i \in \mathbb{Z}_{\geq 0}^d$ and a vector $b \in int.cone(S \cap \mathbb{Z}^d)$. We consider integral points in $cone(B)$ generated by the vertices $B = \{B_0, B_1, \ldots, B_d\} = V_I$ of the simplex S. For convenience, we denote by B also the matrix with columns $B_0, B_1, \ldots, B_d$. As our main subject of investigation, we consider the parallelepiped

$$\Pi = \{x_0 B_0 + x_1 B_1 + \ldots + x_d B_d \mid x_i \in [0, 1]\}.$$

By definition of $S = \{x_0 B_0 + x_1 B_1 + \ldots + x_d B_d \mid x_i \in [0, 1], \sum_i x_i = 1\}$ we have that $S \subset \Pi$. Furthermore, one can easily see that $cone(B)$ can be partitioned into parallelepipedes Π (see figure 5.1 with $B_0 = 0$), as each point $b = x_0 B_0 + x_1 B_1 + \ldots + x_d B_d \in cone(B) \cap \mathbb{Z}^d$ for some $x \in \mathbb{R}_{\geq 0}^d$ can be written as the sum of an integral part $Bx^{int} = \lfloor x_0 \rfloor B_0 + \ldots + \lfloor x_d \rfloor B_d$ and a fractional part $[Bx] = \{x_0\} B_0 + \ldots + \{x_d\} B_d \in \Pi$ (we denote the fractional part of some $v \in \mathbb{R}$ by $\{v\} = v - \lfloor v \rfloor$ and for some vector $x \in \mathbb{R}^d$ we denote by $\{x\}$ the vector $(\{x_0\}, \ldots, \{x_d\})^T$). Furthermore, let $[b] = [Bx]$.

For the proof of the main theorem, we consider a $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$ with $b = \sum_{s \in S} \lambda_s s$ and suppose that λ does not fulfill property (1) of Theorem 5.3. Then there exists a

$\gamma \in (\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$ with big weight i.e. $\lambda_\gamma \geq 2^{2^{\Omega(d)}}$. The key idea of the proof is that we consider the set of multiplicities $\gamma, 2\gamma, 3\gamma, \ldots$ of the vector γ. Our goal is to find a possibly small multiplicity $K > 1$ such that $K\gamma$ is equivalent to a point δ in the convex hull S. Hence, weight on γ can be shifted to the vertices $B_0, \ldots, B_d$ of S. Then $K\gamma$ can be written as the sum of vertices $\sum \Lambda_i B_i$ plus some $\delta \in S \cap \mathbb{Z}^d$ (see Lemma 5.8 for a detailed proof). In figure 5.1 we have that 3γ is equivalent to a point in the simplex (as remarked by the grey areas) and hence in that case $3\gamma = B_1 + B_2$ for some $\delta \in S$. Before we are ready to prove the existence of a small multiplicity K, we give some definitions and observations.

Instead of multiplicities of $\gamma \in S \cap \mathbb{Z}^d$, we consider multiplicities of a vector $x \in [0,1)^{d+1}$ in the unit cube with $\gamma = x_0 B_0 + x_1 B_1 + \ldots + x_d B_d = Bx$.

Definition 5.7. *Consider multiplicities $x, 2x, 3x, \ldots$ of a vector $x \in [0,1)^{d+1}$ with $\sum x_i = 1$. We say components i* jumps *at K if $\lceil Kx_i \rceil > \lceil (K-1)x_i \rceil$. We define*

$$Level(Kx) = \sum_{i=0}^{d} \{Kx_i\}.$$

The following lemma shows that we obtain the desired decomposition of $K\gamma$ if $Level(Kx) = 1$.

Lemma 5.8. *Let $\gamma \in S \cap \mathbb{Z}^d$ be the vector with $\gamma = Bx$ for $x \in [0,1)^{d+1}$. If $Level(Kx) = 1$, then there exists a $\Lambda \in \mathbb{Z}_{\geq 0}^{d+1}$ and a $\delta \in S \cap \mathbb{Z}^d$ such that*

$$K\gamma = \delta + \sum_{i=0}^{d} \Lambda_i B_i.$$

Proof. As above, we split every component i of $Kx \in \mathbb{R}_{\geq 0}^{d+1}$ into an integral part $Kx_i^{int} = \lfloor Kx_i \rfloor$ and a fractional part $\{Kx_i\}$. Then $Kx = Kx^{int} + \{Kx\}$ and we set $\delta = B(\{Kx\}) = [B(Kx)]$.

Observation 5.9. $\delta \in \mathbb{Z}^d$

Since $K\gamma = KBx \in \mathbb{Z}^d$ and $KBx^{int} \in \mathbb{Z}^d$ we obtain that $\delta = B(\{Kx\}) = B(Kx) - B(Kx^{int})$ is integral and therefore $\delta \in \mathbb{Z}^d$.

Observation 5.10. $\delta \in S$*:*

Since $Level(Kx) = 1$ we obtain that $\sum_{i=0}^{d} \{Kx_i\} = 1$ and $\delta = B(\{Kx\}) = (\{Kx_0\})B_0 + \ldots + (\{Kx_d\})B_d$, we can state δ as a convex combination of $B_0, B_1, \ldots B_d$. Therefore $\delta \in S$.

Finally, we can decompose $K\gamma$ into

$$K\gamma = KBx = B(\{Kx_d\}) + B(Kx^{int}) = B(Kx^{int}) + \delta = \delta + \sum_{i=0}^{d} \Lambda_i B_i$$

for some $\Lambda \in \mathbb{Z}_{\geq 0}^{d+1}$. □

The following lemma gives a correlation between the level of some point Kx and the number of jumps.

Lemma 5.11. *Let J be the number of jumps at K, then*

$$Level(Kx) = Level((K-1)x) + 1 - J.$$

Proof. For every component i of Kx which jumps, we obtain that $\{Kx_i\} = \{(K-1)x_i + x_i\} = \{(K-1)x_i\} + x_i - 1$. Hence

$$\begin{aligned}
&Level(Kx) = \sum_{i=0}^{d} \{Kx_i\} \\
&= \sum_{i \text{ jumps at } K} (\{(K-1)x_i\} + x_i - 1) + \sum_{i \text{ does not jump at } K} (\{(K-1)x_i\} + x_i) \\
&= \sum_{i=0}^{d} \{(K-1)x_i\} + \sum_{i=0}^{d} x_i - J = Level((K-1)x) + 1 - J.
\end{aligned}$$

□

Theorem 5.12. *Given $x \in [0,1)^{d+1}$ with $Level(x) = \sum_{i=0}^{d} x_i = 1$. Then there is a $K \in \mathbb{N}_{>1}$ with $K \leq 2^{2^{O(d)}}$ such that $Level(Kx) = \sum_{i=0}^{d} \{Kx_i\} = 1$.*

Proof. We suppose that $\frac{1}{2} > x_0 \geq x_1 \geq \ldots \geq x_d$. In the case that $x_0 \geq \frac{1}{2}$ we obtain that $Level(2x) \leq 1$ and are done. Let $\bar{d} \leq d$ be the smallest index such that

$$\sum_{i=\bar{d}+1}^{d} x_i < \frac{1}{X(\bar{d})} x_{\bar{d}},$$

where $X(\bar{d}) = \prod_{i=0}^{\bar{d}} p_i$ and $p_i = \lceil \frac{1}{x_i} \rceil$. Intuitively, $\bar{d}$ is chosen such that there is a major jump from $x_{\bar{d}}$ to $x_{\bar{d}+1}$, i.e. $x_{\bar{d}} >> x_{\bar{d}+1}$. Note that the above equation is always fulfilled for $\bar{d} = d$ and since $\frac{1}{X(\bar{d})} x_{\bar{d}} < x_{\bar{d}} < 1/2$ we have that $\bar{d} \geq 1$. First, we prove the following lemma to give bounds for $X(\bar{d})$ and $x_{\bar{d}}$

Lemma 5.13. *Assuming for every $0 \leq j \leq \bar{d}$ that $\sum_{i=j+1}^{d} x_i \geq \frac{1}{X(j)} x_j$, then the following parameters can be bounded by*

- *$X(\bar{d}) \leq 2^{2^{O(d)}}$ and*
- *component $x_{\bar{d}} \geq \frac{1}{2^{2^{\Omega(d)}}}$.*

Proof. For $j = 0$ we know that $x_0 \geq \frac{1}{d+1}$ as x_0 is the largest component. This implies also that $X(0) \leq d + 1$. We suppose by induction that for every $j \leq \bar{d}$,

$$x_j \geq (2^{j2^{2j}} \cdot d^{2^{2j}})^{-1}$$

and

$$X(j) \leq (2^{j2^{2j+1}} \cdot d^{2^{2j+1}}).$$

Since $j \leq \bar{d}$ we obtain that $\sum_{i=j+1}^{d} x_i \geq \frac{1}{X(j)} x_j$ and since the coefficients are sorted in non-increasing order we get $d x_{j+1} \geq \frac{1}{X(j)} x_j$. Using the induction hypothesis this gives

$$\begin{aligned}
x_{j+1} &\geq (2^{j2^{2j+1}} d^{2^{2j+1}})^{-1} \cdot (2^{j2^{2j}} d^{2^{2j}})^{-1} \cdot d^{-1} \\
&\geq (2^{j2^{2j+1} + j2^{2j}} \cdot d^{2^{2j} + 2^{2j+1} + 1})^{-1} \\
&> (2^{2 \cdot j2^{2j+1}} \cdot d^{2 \cdot 2^{2j+1}})^{-1} \\
&= (2^{j2^{2(j+1)}} \cdot d^{2^{2(j+1)}})^{-1}
\end{aligned}$$

Product $X(j+1)$ can be bounded as follows:

$$\begin{aligned}
X(j+1) &= \lceil \frac{1}{x_{j+1}} \rceil X(j) \leq (\frac{1}{x_{j+1}} + 1)X(j) \\
&\leq (2^{(j+1)2^{2(j+1)}} d^{2^{2(j+1)}} + 1) \cdot 2^{j2^{2j+1}} d^{2^{2j+1}} \\
&< 2^{(j+1)2^{2(j+1)}+1} d^{2^{2(j+1)}} \cdot 2^{j2^{2j+1}} d^{2^{2j+1}} \\
&= 2^{(j+1)2^{2(j+1)}+1+j2^{2j+1}} \cdot d^{2^{2(j+1)}+2^{2j+1}} \\
&< 2^{2\cdot(j+1)2^{2(j+1)}} \cdot d^{2\cdot 2^{2(j+1)}} \\
&= 2^{(j+1)2^{2(j+1)+1}} \cdot d^{2^{2(j+1)+1}}
\end{aligned}$$

As a result we obtain that $X(\bar{d}) \leq (2^{\bar{d}2^{2\bar{d}+1}} \cdot d^{2^{2\bar{d}+1}}) = 2^{2^{O(d)}}$ and $x_{\bar{d}} \geq (2^{\bar{d}2^{2\bar{d}}} \cdot d^{2^{2\bar{d}}})^{-1} = \frac{1}{2^{2^{\Omega(d)}}}$. □

Let $X = X(\bar{d})$, for each component $0 \leq i \leq d$, we define the *distance* $D_i(K)$ of a multiplicity K by $D_i(K) = j$, where $j \geq 0$ is the smallest integer such that component i jumps at $K + j$. Note that $D_i(K)$ is bounded by p_i as $p_i x_i \geq 1$. We say $Kx \equiv K'x$ if for every $0 \leq i \leq \bar{d}$ the distance $D_i(K) = D_i(K')$. Consider elements $x, 2x, \ldots, (X+1)x$. Since the number of equivalence classes is bounded by $X = \prod_{i=1}^{\bar{d}} p_i$, there exist two elements $Kx, (K+Z)x$ with $K, Z \in \mathbb{Z}_{\geq 1}$ and $K, (K+Z) \leq X+1$ such that $Kx \equiv (K+Z)x$. We will see that the equivalence of two multiplicities implies the existence of a multiplicity $M > 1$ with $Level(Mx) = 1$.

First, we argue about the level of multiplicity $Z-1$. Note that since $\sum_{i=0}^{d} x_i = 1$ we have that $Level(Kx) = \sum_{i=0}^{d} \{Kx_i\}$ for every multiplicity $K \in \mathbb{Z}_{\geq 1}$. The case that $Level((Z-1)x) \geq \bar{d} + 2$ is not possible since

$$\begin{aligned}
Level((Z-1)x) &= \sum_{i=0}^{d} \{(Z-1)x_i\} \leq \bar{d} + 1 + (Z-1) \sum_{i=\bar{d}+1}^{d} x_i \leq \bar{d} + 1 + (X-1) \sum_{i=\bar{d}+1}^{d} x_i \\
&\leq \bar{d} + 1 + \frac{X-1}{X} x_{\bar{d}} < \bar{d} + 2.
\end{aligned}$$

Case (1)**.** *Suppose* $Level((Z-1)x) = \bar{d} + 1$ *(for* $Z \geq 2$*).*

Case (1a)**.** *Suppose* $\{(Z-1)x_i\} \geq 1 - x_i$ *for all* $i = 0, 1, \ldots, \bar{d}$*.*

In this case every component $0 \leq i \leq \bar{d}$ jumps at Z. By Lemma 5.11, we can bound the level of Zx by $Level(Zx) \leq Level((Z-1)x) + 1 - (\bar{d} + 1) = 1$.

Case (1b)**.** *There is an* $0 \leq i \leq \bar{d}$ *such that* $\{(Z-1)x_i\} < 1 - x_i$*.*

In this case $\sum_{i=0}^{\bar{d}} \{(Z-1)x_i\} \leq \bar{d} + 1 - x_i \leq \bar{d} + 1 - x_{\bar{d}}$ and we obtain

$$\begin{aligned}
Level((Z-1)x) &= \sum_{i=0}^{d} \{(Z-1)x_i\} < \bar{d} + 1 - x_{\bar{d}} + \sum_{i=\bar{d}+1}^{d} (Z-1)x_i \leq \bar{d} + 1 - x_{\bar{d}} + \frac{1}{X} x_{\bar{d}} \\
&< \bar{d} + 1
\end{aligned}$$

which contradicts the assumption of Case 1.

Case (2)**.** *Suppose* $Level((Z-1)x) \leq \bar{d}$*.*

Since $K \equiv K + Z$ we know that every component i jumps at $K + D_i$ and $K + Z + D_i$, hence for every $0 \leq i \leq \bar{d}$ we obtain $\{(K + D_i)x_i\} < x_i$ and $\{(K + Z + D_i)x_i\} < x_i$. Let $\{(K + D_i)x_i\} = \alpha_1$ and $\{(K + Z + D_i)x_i\} = \alpha_2$ for some $\alpha_1, \alpha_2 < x_i$. Then $\{Zx_i\} = \{\alpha_2 - \alpha_1\}$ and hence $\{Zx_i\} = \alpha_2 - \alpha_1$ if $\alpha_1 \leq \alpha_2$ and $\{Zx_i\} = 1 + \alpha_2 - \alpha_1$ if $\alpha_1 > \alpha_2$. Since $\alpha_1, \alpha_2 < x_i$ we have $\{Zx_i\} < x_i$ or $\{Zx_i\} \geq 1 - x_i$. Hence every component $0 \leq i \leq \bar{d}$ jumps at Zx_i or at $(Z + 1)x_i$. We obtain by Lemma 5.11 that $Level((Z + 1)x) \leq Level((Z - 1)x) + 2 - (\bar{d} + 1) \leq \bar{d} + 2 - (\bar{d} + 1) = 1$. □

Proof of the main Theorem 5.3

Consider the vector $b \in int.cone(S \cap \mathbb{Z}^d)$. Let $\lambda \in \mathbb{Z}^{S \cap \mathbb{Z}^d}$ be the integral vector with $b = \sum_{s \in S \cap \mathbb{Z}^d} \lambda_s s$.

Assume there is a component $\gamma \in (S \cap \mathbb{Z}^d) \setminus B$ with high multiplicity i.e. $\lambda_\gamma = 2^{2^{\Omega(d)}}$. Since $\gamma \in S$, there is a $x \in \mathbb{R}^{d+1}_{\geq 0}$ with $\sum_{i=0}^{d} x_i = 1$ and $\gamma = x_0 B_0 + x_1 B_1 + \ldots x_d B_d$. By Theorem 5.12 there exists a multiplicity $K = 2^{2^{O(d)}} > 1$ such that $Level(Kx) = 1$. According to Lemma 5.8, there exists a $\delta \in S \cap \mathbb{Z}^d_{\geq 0}$ such that $K\gamma = \delta + \sum_{i=0}^{d} \Lambda_i B_i$ for some $\Lambda \in \mathbb{Z}^{d+1}_{\geq 0}$. Then we can construct a $\lambda' \in \mathbb{Z}^{\bar{d}+1}_{\geq 0}$ with $b = \sum_{s \in S \cap \mathbb{Z}^d} \lambda'_p p$, which has more weight in B as $K > 1$:

$$\lambda' = \begin{cases} \lambda_\gamma - K & \\ \lambda_\delta + 1 & \\ \lambda_{B_i} + \Lambda_i & \forall B_i \in B \\ \lambda_s & \forall s \in (S \cap \mathbb{Z}^d) \setminus (B \cup \{\gamma, \delta\}) \end{cases}$$

Since $K > 1$, the resulting vector λ' has a decreased vertex distance as the sum $\sum_{p \in (\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I} \lambda_p$ is reduced at least by 1. Using Theorem 5.1 applied to components $i \in (S \cap \mathbb{Z}^d) \setminus B$, we can construct a $\lambda^{(1)}$ with $|supp(\lambda'' \setminus B)| \leq 2^d$. In the case that there is another component $\gamma \in (S \cap \mathbb{Z}^d) \setminus B$ with multiplicity $\lambda''_\gamma = 2^{2^{\Omega(d)}}$ we can iterate this process. In the other case, solution λ'' fulfills the proposed properties.

Proof of the structure Theorem 5.5

Proof. Given polytope $\mathcal{P} = \{x \in \mathbb{R}^d \mid Ax \leq c\}$ for some matrix $A \in \mathbb{Z}^{m \times d}$ and a vector $c \in \mathbb{Z}^d$ and let $\mathcal{P}_I$ be the integer polytope with vertices V_I. The structure theorem follows easily by decomposing the polytope $\mathcal{P}$ into simplices S of the form $S = Conv(B_0, B_1, \ldots B_d)$ for $B_i \in V_i$.

By Caratheodory's Theorem, there exist for each $\gamma \in \mathcal{P}$ vertices $B_0, B_1, \ldots, B_d \in V_I$ and a $x \in \mathbb{R}^{d+1}_{\geq 0}$ with $\sum_{i=0}^{d} x_i = 1$ such that $\gamma = x_0 B_0 + \ldots + x_d B_d$. Consider the vector $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$. Let $\lambda \in \mathbb{Z}^{\mathcal{P} \cap \mathbb{Z}^d}$ be the integral vector with $b = \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p$. By Theorem 5.1, we can assume that $supp(\lambda) \leq 2^d$ and hence there are at most 2^d simplices $S^{(k)} = Conv(B_0^1, B_1^{(k)}, \ldots B_d^{(k)})$ for $k = 1, \ldots, 2^d$ which contain a point $\gamma \in \mathcal{P}$ with $\lambda_\gamma > 0$. Finally, we can apply our main Theorem 5.3 to every simplex $S^{(k)}$ for $k = 1, \ldots, 2^d$ to obtain a vector $\lambda' \in \mathbb{Z}^{\mathcal{P} \cap \mathbb{Z}^d}_{\geq 0}$ which fulfills the above properties. □

5.2.1 Algorithmic application

Computing V_I in fpt-time

Given Polytope $\mathcal{P} = \{x \in \mathbb{R}^d \mid Ax \leq c\}$ for some matrix $A \in \mathbb{Z}^{m \times d}$ and a vector $c \in \mathbb{Z}^d$ such that all coefficients of A and c are bounded by Δ. Cook et al. [Coo+92] proved that the number of vertices V_I of the integer polytope $\mathcal{P}_I$ is bounded by $m^d \cdot O((\log \Delta))^d$. In the following we give a brief description on how the set of vertices V_I can be computed in time $|V_I| \cdot d^{O(d)} \cdot (m \log(\Delta))^{O(1)}$ and therefore in fpt-time parameterized by the number of vertices $|V_I|$. For a detailed description of the algorithm we refer to the thesis of Hartmann [Har89].

Given at timestep t a set of vertices $V_t \subset V_I$ and the set of facets $F^{(t)} = \{F_1, \ldots, F_\ell\}$ of $conv(V_t)$ corresponding to half-spaces $H_i = \{x \mid n_i x \leq c_i\}$ for normal vectors $n_1, \ldots, n_\ell \in \mathbb{R}^d$ and constants $c_1, \ldots, c_d \in \mathbb{Z}$ with $conv(V) = \cap_{i=1}^{\ell} H_i$. Consider for every $1 \leq i \leq \ell$ the polytope $\mathcal{P} \cap H_i^-$ for a halfspace $H_i^- = \{x \mid x \in \mathcal{P}, n_i x \geq c_i\}$. Compute with Lenstra's algorithm [LJ83] a solution $x^* \in \mathcal{P} \cap \mathbb{Z}^d$ of the ILP $\max\{n_i x \mid x \in (\mathcal{P} \cap \mathbb{Z}^d), n_i x \geq c_i\}$. In the case that $n_i x^* > c_i$, solution x^* does not belong to $Conv(V_t)$. Assuming that x^* is a vertex of the integer polytope of $\mathcal{P} \cap H_i^-$, solution x^* is also a vertex of the integer polytope $\mathcal{P}_I$. We can add x^* to the set of existing vertices $V_t \subset V_I$, construct the increased set of facets $F^{(t+1)}$ of $Conv(V_t \cup \{x^*\})$ and iterate the procedure. In the case that there is no solution x^* with $n_i x^* > c_i$ for any $1 \leq i \leq \ell$, we have that $\mathcal{P}_I = Conv(V_t)$ and are done.

Bin Packing in fpt-time

In the following we describe the algorithmic use of the presented structure theorem 5.5. Therefore, we follow the approach by Goemans and Rothvoß [GR14].

Theorem 5.14. *Given polytopes $\mathcal{P}, \mathcal{Q} \subset \mathbb{R}^d$, one can find a $y \in int.cone(\mathcal{P} \cap \mathbb{Z}^d) \cap \mathcal{Q}$ and a vector $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$ such that $b = \sum_{p \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p$ in time $|V_I|^{2^{O(d)}} enc(\mathcal{P})^{O(1)} enc(\mathcal{Q})^{O(1)}$, where $enc(\mathcal{P}), enc(\mathcal{Q})$ is the encoding length of the polytope $\mathcal{P}, \mathcal{Q}$ or decide that no such y exists.*

Proof. Let $\mathcal{P} = \{x \in \mathbb{R}^d \mid Ax \leq c\}$ and $\mathcal{Q} = \{x \in \mathbb{R}^d \mid \tilde{A}x \leq \tilde{c}\}$ be the given polytopes for a matrix $A \in \mathbb{Z}^{m \times d}$ and a matrix $\tilde{A} \in \mathbb{Z}^{\tilde{m} \times \tilde{d}}$. First, we compute the set of vertices of $\mathcal{P}_I$ in time $|V_I|^{2^{O(d)}} enc(\mathcal{P})^{O(1)} enc(\mathcal{Q})^{O(1)}$ as described above. Suppose that there is a vector $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d) \cap \mathcal{Q}$, then by Theorem 5.5, we know there is a vector $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$ with $b = \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_s s$ such that

1. $\lambda_p \leq 2^{2^{O(d)}} \forall p \in (\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$,
2. $|supp(\lambda) \cap V_I| \leq d \cdot 2^d$,
3. $|supp(\lambda) \setminus V_I| \leq 2^{2d}$.

At the expense of a factor $\binom{|V_I|}{d2^d} = |V_I|^{2^{O(d)}}$ we can guess the support $V_\lambda \subseteq V_I$ of λ restricted to components λ_p with $p \in V_I$ i.e. $V_\lambda = supp(\lambda) \cap V_I$. For each $p \in V_\lambda$ we use variables $\bar{\lambda}_p \in \mathbb{Z}_{\geq 0}$ to determine the multiplicities of λ_p. Furthermore, we guess the number of different points $p \notin V_i$ used in λ i.e. $k = |supp(\lambda) \setminus V_I| \leq 2^{2d}$. We use variables $x_i^{(j)}$ for $j = 1, \ldots, k$ to determine the points $p \notin V_I$ and their multiplicity λ_p. Note that in the following ILP, we encode the multiplicity of a λ_p with $p \notin V_I$ binary, therefore the number

of variables $x_i^{(j)}$ can be bounded by $2^{2d} \cdot \log(2^{2^{O(d)}}) = 2^{O(d)}$. And finally we use a vector $y \in \mathbb{Z}^d$ to denote the target vector in polytope $\mathcal{Q}$.

$$
\begin{aligned}
A x_i^{(j)} &\leq b \qquad \forall i = 1, \ldots, k \text{ and } \forall j = 1, \ldots, k \\
\sum_{p \in V_\lambda} \bar{\lambda}_p p + \sum_{j=1}^{k} \sum_{i=1}^{2^{O(d)}} 2^j x_i^{(j)} &= y \\
\tilde{A} y &\leq \tilde{b} \\
x_i &\in \mathbb{Z}^d \qquad i = 1, \ldots, k \\
\bar{\lambda}_p &\in \mathbb{Z}_{\geq 0} \qquad \forall p \in V_\lambda \\
y &\in \mathbb{Z}^d
\end{aligned}
$$

Using the algorithm of Lenstra or Kannan ([LJ83],[Kan87]) to solve the above ILP which has $k2^{O(d)}+d+d2^d = 2^{O(d)}$ variables and $mk+d+\tilde{m}+d|V_\lambda| = m2^{O(d)}+\tilde{m}$ constraints, takes time $(2^{O(d)})^{2^{O(d)}} \cdot (m2^{O(d)} + \tilde{m})^{O(1)} \log(\bar{\Delta})^{O(1)} = 2^{2^{O(d)}} enc(\mathcal{P})^{O(1)} enc(\mathcal{Q})^{O(1)}$, where $\bar{\Delta} = \max\{2^{2^{O(d)}}, d!\Delta^d, \tilde{\Delta}\}$. We obtain for the total running time: $|V_I|^{2^{O(d)}} enc(\mathcal{P})^{O(1)} enc(\mathcal{Q})^{O(1)}$ □

We can apply this theorem to the bin packing problem by choosing $\mathcal{P} = \{\begin{pmatrix} x \\ 1 \end{pmatrix} \in \mathbb{R}^{d+1}_{\geq 0} \mid s^T x \leq 1\}$ and $\mathcal{Q} = \{b\} \times [0, a]$ to decide if items b can be packed into at most a bins. Using binary search, on the number of used bins, we can solve the bin packing problem in time $|V_I|^{2^{O(d)}} \cdot \log(\Delta)^{O(1)}$, where Δ is the largest multiplicity of item sizes or the largest denominator appearing in an itemsize $s_1, \ldots, s_d$. Since $|V_I| \geq d + 1$ this running time is fpt-time, parametrized by the number of vertices $|V_I|$ and therefore we obtain Theorem 5.6.

5.3 Lower Bound

In this section we give a construction of a bin packing instance (s, b) with vertex distance $Dist(b) = 2^{2^{\Omega(d)}}$. We consider the case that all item sizes $s_1, \ldots, s_d$ are of the form $s_i = \frac{1}{a_i}$ for some $a_i \in \mathbb{Z}_{\geq 1}$. In this case, all edges of the knapsack polytope $\mathcal{P} = \{x \in \mathbb{R}^{\geq 0} \mid s_1 x_1 + \ldots + s_d x_d \leq 1\}$ are of the form $B_i = (0, \ldots, 0, a_i, 0, \ldots, 0)^T$ and therefore integral. We obtain that $\mathcal{P}_I = \mathcal{P} = Conv(0, B_1, \ldots, B_d)$.

Our approach for the proof of the lower bound is as follows: We prove the existence of a parallelepipped $\Pi = \{x_1 B_1 + \ldots + x_d B_d \mid x_i \in [0, 1]\}$ with a special element $g \in (\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$ such that the unique optimal packing of the bin packing instance $K \cdot g$ (for a possibly large multiplicity $K \in \mathbb{Z}_{\geq 0}$) is to use K times the configuration g. In this case, weight can not be shifted to the vertices $B_1, \ldots, B_d$ and hence the instance Kg implies a vertex distance of $Dist(Kg) = K$. We show that the special element g can be determined by a set of modulo congruences. Therefore, we are able to use basic number theory to construct a bin packing instance with double exponential vertex distance.

First, we take a close look at the parallelepipped Π and $Cone(B)$. We say that two points $b, b' \in Cone(B)$ are *equivalent* if $[b] = [b']$. Each point $b \in Cone(B)$ is equivalent to a point in the parallelepiped Π.

Lemma 5.15. *Using operation $+$ defined by $p + p' = [p + p']$ for some $p, p' \in \Pi \cap \mathbb{Z}^d$ then $G(\Pi) = (\Pi \cap \mathbb{Z}^d, +)$ is an abelian group with $|G(\Pi)| = |det(B)|$ many elements.*

The proof that $G(\Pi)$ is a group can be easily seen, since $G(\Pi)$ is the quotient group of $\mathbb{Z}^d$ and the lattice $\mathbb{Z}B_1 \oplus \ldots \oplus \mathbb{Z}B_d$. For the fact that $|G(\Pi)| = det(B)$ we refer to [Bar07]. In the considered case that $B_i = (0, \ldots, 0, a_i, 0, \ldots, 0)^T$, the group $G(\Pi)$ is isomorphic to $(\mathbb{Z}/a_1\mathbb{Z}) \times \ldots \times (\mathbb{Z}/a_d\mathbb{Z})$. Recall that the set $\mathcal{P} \cap \mathbb{Z}^d \subset \Pi \cap \mathbb{Z}^d$ represents all integral points of the knapsack polytope and each $\delta \in \mathcal{P} \cap \mathbb{Z}^d$ therefore represents a way of packing a bin with items from sizes $s_1, \ldots, s_d$. We call $\delta \in \mathcal{P} \cap \mathbb{Z}^d$ a *configuration*.

In the following subsection 5.3.3, we also give an easy observation on how the vertex distance is connected to the integrality gap of the bin packing problem.

5.3.1 Preliminaries

In this section we state some basic number theoretic theorems that we will use in the following. For details and proofs, we refer to the books of Stark [Sta70] and Graham, Knuth and Patashnik [GKP94].

Theorem 5.16 ([Sta70])**.** *Let $a_1, \ldots a_d \in \mathbb{Z}$ with $gcd(a_1, \ldots, a_d) = 1$, then there exist $v_1, \ldots, v_d \in \mathbb{Z}$ such that*

$$a_1 v_1 + \ldots a_d v_d = 1.$$

Theorem 5.17 (Chinese remainder theorem [Sta70])**.** *Suppose $a_1, \ldots, a_d \in \mathbb{Z}$ are pairwise coprime. Then, for any given sequence of integers $i_1, \ldots, i_d$, there exists an integer x solving the following system of simultaneous congruences.*

$$x \equiv i_j \mod a_j \text{ for } 1 \leq j \leq d.$$

Furthermore, x is unique $\mod \prod_{i=1}^{d} a_i$.

Theorem 5.18 ([Sta70])**.** *Given congruence $x \mod a$. If x and a are coprime, there exists an inverse element $x^{-1} \in \mathbb{Z}/a\mathbb{Z}$ such that $xx^{-1} \equiv 1 \mod a$.*

Sylvester's sequence is defined by,

$$S_1 = 2$$
$$S_j = 1 + \prod_{i=1}^{j-1} S_i$$

and has following properties (see [GKP94])

$$S_n \approx 1.264^{2^n}$$
$$\sum_{i=1}^{j-1} \frac{1}{S_i} = 1 - \frac{1}{S_j - 1}$$

5.3.2 Proof of the lower bound

We start by defining the *size* of an element $\pi \in \Pi$ by

$$Size(\pi) = \sum_{i=1}^{d} s_i \pi_i.$$

In our case, the sizes s_i are given by $s_i = \frac{1}{a_i}$ and vectors $B_i = (0, \ldots, 0, a_i, 0, \ldots, 0)^T$ for some $a_i \in \mathbb{Z}_{\geq 1}$. Hence, the size of a B_i equals to 1 and for each $x \in [0,1)^d$ with $Bx = \pi$, we have that $\sum_{i=1}^d x_i = Size(\pi)$. Since the matrix B is a diagonal matrix with entries a_i, the determinant equals $det(B) = \prod_{i=1}^d a_i$. We define for $1 \leq i \leq d$ that

$$R_i = \frac{det(B)}{a_i} = \prod_{j \neq i} a_j.$$

In the following lemma we show that the fractional value of the size $\{Size(\Pi)\}$ is unique for every element $\pi \in G(\Pi)$.

Lemma 5.19. *Given parallelepiped $\Pi = \{x_1 B_1 + \ldots + x_d B_d \mid x_i \in [0,1)^d\}$ with $B_i = (0, \ldots, 0, a_i, 0, \ldots, 0)^T$. If $a_1, \ldots, a_d$ are pairwise coprime, then for every $0 \leq a < det(B)$, there exists a unique vector $\pi \in \Pi \cap \mathbb{Z}^d$ and a vector $x \in [0,1)^d$ with $Bx = \pi$, such that*

$$Size(\pi) = \sum_{i=1}^d x_i = z + a/det(B),$$

for some $z \in \mathbb{Z}_{\geq 0}$.

Proof. Since $a_1 \ldots, a_d$ are pairwise coprime, we have that $gcd(R_i, R_{i+1}) = \prod_{j \neq i, i+1} a_j$ and hence $gcd(R_1, \ldots, R_d) = 1$. By Theorem 5.16, there exist $v_1, \ldots, v_d \in \mathbb{Z}$ such that $v_1 R_1 + \ldots + v_d R_d = 1$. For $v_i' = v_i R_i \mod det(B)$ the sum $\sum_{i=1}^d v_i' \equiv 1 \mod det(B)$. Consider the vector $x = (\frac{v_1'}{det(B)}, \ldots, \frac{v_d'}{det(B)})^T$, then $\sum_{i=1}^d x_i = \sum_{i=1}^d \frac{v_i'}{det(B)} = \frac{1}{det(B)} + z$ for some $z \in \mathbb{Z}_{\geq 0}$. The vector $Bx = \frac{v_1'}{det(B)} B_1 + \ldots + \frac{v_d'}{det(B)} B_d$ is integral since for every $1 \leq i \leq d$ the congruence $v_i' a_i \equiv v_i R_i a_i \equiv v_i det(B) \equiv 0 \mod det(B)$ holds and hence each item sizes $a_i x_i = \pi_i = \frac{v_i'}{det(B)} a_i \in \mathbb{Z}_{\geq 0}$.

Consider multiplicities $x, 2x, 3x, \ldots$. As above we can rewrite each element $Kx \in \mathbb{R}^d_{\geq 0}$ by $Kx = Kx^{int} + \{Kx\}$ with $\{Kx\} = (\{Kx_1\}, \ldots, \{Kx_d\})^T$ and $\{Kx_i\} < 1$, which implies that $B(\{Kx\}) \in \Pi$. Since $B(Kx)$ is integral and $B(Kx^{int})$ is integral, the vector $B(\{Kx\})$ is integral as well. Furthermore, the sum of all component $\{Kx\}$ sums up to $\sum_{i=1}^d \{Kx_i\} = K \sum_{i=1}^d x_i - \sum_{i=1}^d \lfloor Kx_i \rfloor = \frac{K}{det(B)} + z$, for some $z \in \mathbb{Z}_{\geq 0}$. Hence for each multiplicity Kx in $0x, x, 2x, \ldots, (det(B)-1)x$ there is a vector $B(\{Kx\}) \in \Pi$ with $\sum_{i=1}^d \{Kx_i\} = z + \frac{K}{det(B)}$ and since Π contains exactly $det(B)$ many elements (see Lemma 5.15), each element of $G(\Pi)$ corresponds to a unique element of $0x, x, 2x, \ldots, (det(B)-1)x$. □

Consider the specific element $g \in \Pi \cap \mathbb{Z}^d$ with fractional vector $x \in [0,1)^d$ such that $Bx = g$ and $Size(g) = \frac{det(B)-1}{det(B)} + z$. We call g the *full generator* of the group $G(\Pi)$.

Corollary 5.20. *For every element $\pi \in G(\Pi)$ there exists a multiplicity K such that $Kg = \pi$, i.e. the full generator g generates the group Π and hence $G(\Pi) =< g >$ is a cyclic group. Element $Kg \in \Pi$ has a size of $z + \frac{det(B)-K}{det(B)}$ for some $z \in \mathbb{Z}_{\geq 0}$.*

Proof. In the proof of the lemma above, we showed that the element Bx with $\sum_{i=1}^d x_i = z + \frac{1}{det(B)}$ generates $G(\Pi)$ as each multiplicity Kx of x yields an element $B(\{Kx\}) \in G(\Pi)$ of size $z + \frac{K}{det(B)}$. We consider the full generator g with $g = Bx'$ for some x' with $\sum_{i=1}^d x_i' = z + \frac{det(B)-1}{det(B)}$ for some $z \in \mathbb{Z}_{\geq 0}$. By the same argument as before, the multiplicities Kx' yield elements $\pi' = B(\{Kx'\}) \in G(\Pi)$ with $Size(\pi') = z + \frac{det(B)-K}{det(B)}$ for some $z \in \mathbb{Z}_{\geq 0}$. □

As above, we consider multiplicities Kg of a vector $g \in \Pi$ and some $K > 0$. We say that $Kg \in cone(B)$ is *unique* if $g \in \mathcal{P}$ and $2g, \ldots, Kg \notin \mathcal{P}$ i.e. g is a configuration and $2g, \ldots, Kg$ are not. In the following lemma we prove that if Kg is unique and g is a full generator, then using K-times configuration g is the unique optimal packing for instance Kg.

Lemma 5.21. *Let g be the full generator of $G(\Pi)$. If $Kg \in cone(B)$ is unique, then there is no $\lambda \in \mathbb{Z}_{\geq 0}^{(\mathcal{P} \cap \mathbb{Z}^d)}$ with $\lambda_g \neq K$ such that $\sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p = Kg$ and $|\lambda| = K$.*

Proof. Consider bin packing instance $Kg \in cone(B)$ and a packing of the instance into bins $1, \ldots, K$. Since g contains items of size $\frac{det(B)-1}{det(B)}$, items in instance Kg have a total size of $K\frac{det(B)-1}{det(B)}$ and therefore, the bins $1, \ldots, K$ have total free space of $\frac{K}{det(B)}$. Each bin configuration $c_1, \ldots, c_K$ of bins $1, \ldots, K$ belongs to $\mathcal{P}$ and hence to $\Pi(G)$. By Cororllary 5.20 for each c_i there exists a multiplicity $K_i \in \mathbb{Z}_{\geq 1}$ such that $K_i g = c_i$. Assuming that $c_i \neq g$ and hence $K_i > 1$ we know that $K_i > K$ as by definition of the uniqueness of Kg, elements $2g, \ldots, Kg$ are no configurations. However, a bin with configuration $K_i g = c_i \in \mathcal{P}$ with $K_i > K$ has free space $\frac{K'}{det(B)} > \frac{K}{det(B)}$ and hence more free space than the total sum of free space in bins $1, \ldots, K$. Therefore, a configuration $\neq g$ can not appear in an optimal packing of the instance Kg. The unique way of packing instance Kg into K bins is to use K times configuration g. □

Consider the full generator $g = x_1 B_1 + \ldots + x_d B_d \in \mathcal{P}$ with $x_i \geq 0$, we say g has the *long-run* property if $(1-\epsilon)\frac{1}{S_i} \leq x_i < \frac{1}{S_i}$ for $1 \leq i \leq d-1$ and some $\epsilon < (\frac{1}{S_d - 1})^2$, where S_i is the i-th sylvester number. The following inequality gives a lower bound for $\|x\|_1$:

$$\begin{aligned}\sum_{i=1}^{d-1} x_i &\geq (1-\epsilon)\sum_{i=1}^{d-1}\frac{1}{S_i} = (1-\epsilon)(1-\frac{1}{S_d-1}) = 1-\epsilon-\frac{1-\epsilon}{S_d-1} \\ &> 1-\frac{1}{(S_d-1)^2}-\frac{1}{S_d-1} = 1-\frac{1}{(S_d-1)^2}+\frac{1}{(S_d-1)(S_d-2)}-\frac{1}{S_d-2} \\ &> 1-\frac{1}{S_d-2}\end{aligned}$$

If g is a configuration and hence $\|x\|_1 \leq 1$, we can bound x_d from above by

$$x_d \leq 1 - \sum_{i=1}^{d-1} x_i < \frac{1}{S_d - 2}$$

Recall that the following statements are equivalent:

- $\{Kg\} \in G(\Pi)$ is a configuration i.e. $Kg \in \mathcal{P}$
- $Level(Kx) = 1$

Lemma 5.22. *If g is a configuration and g has the long run property, then $(S_d - 2)g$ is unique for $d \geq 3$.*

Proof. Let $x \in [0,1]^{d+1}$ such that $x_0 0 + x_1 B_1 + \ldots + x_d B_d = g$ with $\sum_{i=0}^{d} x_i = 1$. We consider the level $Level(Kx)$ of multiplicities of x. Recall that $Level(Kx) = 1$ if and only if $\{Kg\} \in \Pi$ is a configuration. Hence, it remains to prove that that $Level(Kx) > 1$ for every $1 < K \leq S_d - 2$.

By Lemma 5.11 we know that level $Level(Kx) = Level((K-1)x) - J_K + 1$, where J_K is the number of jumps at K. This implies by induction that $Level(Kx) = K - J$, where J is the total sum of all jumps in $2x, \ldots, Kx$. Using that $\sum_{i=1}^{d-1} x_i > 1 - \frac{1}{S_d - 2}$, we obtain that $x_0, x_d < \frac{1}{S_d-2}$ and hence $Kx_0, Kx_d < 1$ for $K \leq S_d - 2$. This means that component 0 and component d do not jump in $2x, \ldots, (S_d - 2)x$.

Observation 5.23. *For every $1 \leq i \leq d-1$, component i jumps at $1 + S_i, 1 + 2S_i, 1 + 3S_i, \ldots, 1 + \lfloor \frac{S_d-2}{S_i} \rfloor S_i$.*

Since $(1-\epsilon)\frac{1}{S_i} \leq x_i < \frac{1}{S_i}$ for $1 \leq i < d$ we know on the one hand that $MS_ix_i < M$ and on the other hand $(1 + MS_i)x_i \geq (1-\epsilon)(\frac{1}{S_i} + M) > M$ as $\epsilon(\frac{1}{S_i} + M) \leq \epsilon(\frac{1+(S_d-2)}{S_i}) < \frac{S_d-1}{(S_d-1)^2}\frac{1}{S_i} < \frac{1}{S_i}$ for $i \leq d-1$ and $M \leq \frac{S_d-2}{S_i}$. Hence component i jumps at $1 + S_i$ from 0 to 1 and at $1 + 2S_i$ from 1 to 2 and so on. The total number of jumps $J_K(i)$ in component i can therefore be bounded by $1 + J_K(i)S_i \leq K$ and hence $J_K(i) \leq \lfloor \frac{K-1}{S_i} \rfloor$.

The total number of jumps J up to $K \leq S_d - 2$ in components $0, \ldots, d$ sums up to

$$J = \sum_{i=0}^{d} \lfloor \frac{K-1}{S_i} \rfloor = \sum_{i=1}^{d-1} \lfloor \frac{K-1}{S_i} \rfloor \leq \lfloor (K-1) \sum_{i=1}^{d-1} \frac{1}{S_i} \rfloor$$

Since $\sum_{i=1}^{d-1} \frac{1}{S_i} = 1 - \frac{1}{S_d - 1}$ we obtain for $K \leq S_d - 2$

$$J \leq \lfloor (K-1) \sum_{i=1}^{d-1} \frac{1}{S_i} \rfloor = \lfloor (K-1)(1 - \frac{1}{S_d - 1}) \rfloor \leq K - 2$$

which implies that $Level(Kx) = K - J \geq K - (K-2) = 2$ and therefore $\{Kg\} \notin \mathcal{P}$ for $K = 2, \ldots, S_d - 2$.

□

Lemma 5.24. *An element $g \in G(\Pi)$ is a full generator if and only if for all $1 \leq i \leq d$*

$$g_i \equiv -R_i^{-1} \mod a_i.$$

Proof. Consider the full generator g of a group $G(\Pi)$. By definition of the full generator, we obtain that there exists a $z \in \mathbb{Z}_{\geq 0}$ such that

$$Size(g) = \sum_{i=1}^{d} s_i g_i = \sum_{i=1}^{d} \frac{g_i}{a_i} = z + \frac{det(B) - 1}{det(B)}$$

and hence

$$det(B) - 1 + z \cdot det(B) = det(B) \sum_{i=1}^{d} \frac{g_i}{a_i} = \sum_{i=1}^{d} R_i g_i$$

By definition of the modulo operation this equation is equivalent to

$$\sum_{i=1}^{d} R_i g_i \equiv det(B) - 1 \mod (det(B)). \tag{5.1}$$

As $det(B) - 1 \equiv -1 \mod a_i$ for each $1 \leq i \leq d$, we obtain by the Chinese remainder Theorem 5.17 (assuming that all a_i's are coprime), that congruence (5.1) is equivalent to the following system of congruences:

$$\sum_{i=1}^{d} R_i g_i \equiv -1 \mod a_i \qquad \text{for } 1 \leq i \leq d$$

As $R_i \equiv 0 \mod a_j$ for any $i \neq j$, we obtain that $\sum_{i=1}^{d} R_i g_i \equiv R_j g_j \mod a_j$ and hence

$$g_i \equiv -R_i^{-1} \mod a_i.$$

□

Sylvester's sequence S_i grows double exponentially by approximately $S_i \approx 1.264^{2^i}$ and therefore $S_i = 2^{2^{\Omega(i)}}$. It remains to prove the existence of sizes $s_1, \ldots, s_d$ with group $G(\Pi)$ such that the full generator of $G(\Pi)$ has the long-run property. The following theorem concludes the proof of a double exponential lower bound.

Theorem 5.4. *There exists a bin packing instance with sizes $\frac{1}{a_1}, \ldots, \frac{1}{a_d}$ for $a_i \in \mathbb{Z}_{\geq 1}$ and multiplicities $b \in \mathbb{Z}^d_{\geq 0}$ corresponding to a point $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$, where $\mathcal{P}$ is the knapsack polytope such that*

$$Dist(b) \geq S_d - 2 = 2^{2^{\Omega(d)}}$$

Proof. Given parallelepiped $\Pi = \{x_1 B_1 + \ldots + x_d B_d \mid x_i \in [0,1)\}$ with configurations $B_i = (0, \ldots, 0, a_i, 0, \ldots, 0)^T$. Assume there are sizes s_i such that group $G(\Pi)$ with full generator $g \in \mathcal{P}$ has the long-run property. Then K times configuration $\begin{pmatrix} 1 \\ g \end{pmatrix}$ is by Lemma 5.21 the unique representation of the vector $b = \begin{pmatrix} K \\ Kg \end{pmatrix} \in int.cone(\mathcal{P}' \cap \mathbb{Z}^d)$ where $\mathcal{P}' = Conv(B'_0, \ldots, B'_d)$ with $B'_0 = (1, 0, \ldots, 0)^T$ and $B'_i = \begin{pmatrix} 1 \\ B_i \end{pmatrix}$. According to Lemma 5.22 this implies a vertex distance of $Dist(b) = S_d - 2 = 2^{2^{\Omega(d)}}$. Therefore, it remains to prove the existence of sizes $s_1, \ldots, s_d$ with group $G(\Pi)$ such that the full generator g of $G(\Pi)$ has the long-run property. In the following we give an inductive construction of the sizes $s_i = \frac{1}{a_i}$:

First, choose a_1 arbitrarily such that there is an m_1 with $(1-\epsilon)\frac{1}{S_1} \leq \frac{m_1}{a_1} < \frac{1}{S_1}$. This is possible for every $a_1 > \frac{S_1}{\epsilon}$ that is not a multiple of $S_1 = 2$. In this case m_1 can be chosen by $m_1 = \lfloor \frac{a_1}{S_1} \rfloor$ and we obtain $\frac{\lfloor a_1/S_1 \rfloor}{a_1} < \frac{1}{S_1}$ and $\frac{\lfloor a_1/S_1 \rfloor}{a_1} \geq \frac{(a_1/S_1)-1}{a_1} \geq \frac{1}{S_1} - \frac{1}{a_1 S_1} \geq (1-\epsilon)S_1$. Additionally, we assume w.l.o.g. that m_1 and a_1 are coprime.

For $1 \leq i < d$ choose a_{i+1} such that there exists an m_{i+1} with $(1-\epsilon)\frac{1}{S_{i+1}} \leq \frac{m_{i+1}}{a_{i+1}} < \frac{1}{S_{i+1}}$. The existence of the m_{i+1} can be shown for any $a_{i+1} > \frac{S_{i+1}}{\epsilon}$ that is not a multiple of S_{i+1} by the same argument as above for m_1. Additionally we choose a_{i+1} such that the following conditions hold:

$$a_{i+1} \equiv (\prod_{j=1}^{i-1} a_j)^{-1} \cdot (-m_i)^{-1} \mod a_i \tag{5.2}$$

$$a_{i+1} \equiv 1 \mod a_j \text{ for } j = 1, \ldots, i-1 \tag{5.3}$$

Remark the following points, where we use the fact that $gcd(a,b) = gcd(a \mod b, b)$ for numbers $a, b \in \mathbb{Z}$.

- The inverse element of $\prod_{j=1}^{i-1} a_j$ and $-m_i$ in $\mathbb{Z}/a_i\mathbb{Z}$ exists since $a_1, \ldots a_j$ are coprime to a_i and m_i is coprime to a_i (see Theorem 5.18),

- since $a_1, \dots a_i$ are coprime, by the chinese remainder theorem 5.17, there exists a unique element $a_{i+1} \mod (\prod_{j=1}^{i} a_j)$ satisfying the above inequalities,
- condition (5.2) implies that a_{i+1} is coprime to a_i as m_i is coprime to a_i and $\prod_{j=1}^{i-1} a_j$ is coprime to a_i (coprimeness carries over to the inverse),
- condition (5.3) implies that a_{i+1} is coprime to $a_1, \dots, a_{i-1}$.

Claim (1)**.** *The full generator g of the constructed group $G(\Pi)$ has the long-run property.*

To prove that $g = x_1 B_1 + \ldots + x_d B_d$ has the long-run property, we show for all $1 \leq i < d$ that $\frac{1}{S_i}(1-\epsilon) \leq \frac{g_i}{a_i} < \frac{1}{S_i}$. By Lemma 5.24

$$g_i \equiv -R_i^{-1} \mod a_i$$

By construction of the a_i we obtain for $g_1, \dots, g_{d-1}$ the following congruences $\mod a_j$:

$$g_i \equiv -\left(\prod_{j=1}^{i-1} a_j \cdot \prod_{j=i+1}^{d} a_j\right)^{-1} \overset{(5.2)}{\equiv} -\left(\prod_{j=1}^{i-1} a_j \cdot a_{i+1}\right)^{-1} \overset{(5.3)}{\equiv} -\left((\prod_{j=1}^{i-1} a_j) \cdot (\prod_{j=1}^{i-1} a_j)^{-1} \cdot (-m_i)^{-1}\right)^{-1}$$
$$\equiv m_i \mod a_i$$

Since for every $\delta \in \Pi$ we have that $\delta_i < a_i$, we know $g_i = m_i$. By definition of m_i we obtain $(1-\epsilon)\frac{1}{S_i} \leq x_i = \frac{m_i}{a_i} < \frac{1}{S_i}$, which proves Claim 1.

Claim (2)**.** *The full generator g is a configuration.*

Suppose $Level(x) > 1$, then we know by Lemma 5.19 that $\sum_{i=1}^{d} x_i = z + \frac{det(B)-1}{det(B)}$ for some $z \in \mathbb{Z}_{\geq 1}$.

$$\sum_{i=1}^{d} x_i < \sum_{i=1}^{d-1} \frac{1}{S_i} + x_d = (1 - \frac{1}{S_d - 1}) + x_d < (1 - \frac{1}{S_d - 1}) + 1$$

Since $x_i = \frac{g_i}{a_i} < \frac{1}{S_i}$ for $1 \leq i < d$, we know that $a_i \geq S_i + 1$ and hence $det(B) > \prod_{i=1}^{d-1} a_i > \prod_{i=1}^{d-1}(S_i + 1) > S_d - 1$ which implies:

$$\sum_{i=1}^{d} x_i < (1 - \frac{1}{det(B)}) + 1 = 1 + \frac{det(B) - 1}{det(B)}$$

This is a contradiction to $Level(x) > 1$. □

5.3.3 Relation between $Dist$ and the IRUP

In this section we study briefly the connection between the vertex distance and the modified integer roundup property (MIRUP) which is defined in the following. Let $\mathcal{P} = \{x \in \mathbb{R}^d_{\geq 0} \mid s^T x \leq 1\}$ be the knapsack polytope for given sizes $s_1, \dots s_d \in (0, 1]$. For given multiplicities $a_1, \dots, a_d$, a packing of the items into a minimum number of bins is given by a solution of the following ILP:

$$\min\{\|\lambda\|_1 \mid \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p = b, \lambda \in \mathbb{Z}^d_{\geq 0}\}. \tag{5.4}$$

The relaxed linear program (LP) is defined by

$$\min\{\|\lambda\|_1 \mid \sum_{p\in\mathcal{P}\cap\mathbb{Z}^d} \lambda_p p = b, \lambda \in \mathbb{R}^d_{\geq 0}\}. \tag{5.5}$$

Let λ^* be an optimal solution of the ILP (5.4) and let λ^f be an optimal solution of the relaxed linear program (5.5), then the integrality gap of an instance (s, b) is defined by:

$$\|\lambda\|_1 - \left\|\lambda^f\right\|_1$$

A well known conjecture by Scheithauer and Terno [ST97] concerning the integrality gap for bin packing instance is that for any instance I, we have that $\|\lambda^*\|_1 \leq \lceil \left\|\lambda^f\right\|_1 \rceil + 1$ which is the so called modified integer roundup property (MIRUP). The integer roundup property (IRUP) is fulfilled if $\|\lambda^*\|_1 \leq \lceil \left\|\lambda^f\right\|_1 \rceil$. In general, bin packing instances where die IRUP is not fulfilled appear rarely. In the literature those kind of instances are studied and constructions of instances are given where die IRUP does not hold (see [ST97], [Cap+14]). In the following we show that a bin packing instance with a large vertex distance $Dist(b)$ implies the existence of many subinstances where the IRUP does not hold. Specifically, we show the following theorem:

Theorem 5.25. *Given a bin packing instance (s, b) corresponding to a vector $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ with vertex distance $Dist(b)$ and let $\lambda \in \mathbb{Z}^{\mathcal{P}\cap\mathbb{Z}^d}$ be a solution with $\sum_{p\in\mathcal{P}\cap\mathbb{Z}^d} \lambda_p p = b$ and vertex distance $\sum_{p\in(\mathcal{P}\cap\mathbb{Z}^d)\setminus V_I} \lambda_p = Dist(b)$. For every $\gamma \in (\mathcal{P}\cap\mathbb{Z}^d)\setminus V_I$ with $\lambda_\gamma = d + Z$ for some $Z \in \mathbb{Z}_{\geq 0}$, there exist at least Z instances where the IRUP does not hold.*

Proof. Given an instance $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ with $Dist(b)$. Then there exists an integral optimal solution $\lambda \in \mathbb{Z}^{\mathcal{P}\cap\mathbb{Z}^d}$ with $\sum_{p\in\mathcal{P}\cap\mathbb{Z}^d} \lambda_p p = b$ and $\sum_{p\in(\mathcal{P}\cap\mathbb{Z}^d)\setminus V_I} \lambda_p = Dist(b)$. We consider for a $\gamma \in (\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$ with $\lambda_\gamma = d + Z$ for some $Z \in \mathbb{Z}_{\geq 1}$ the instances $(d+1)\gamma, \ldots, (d+Z)\gamma$. Let $b' \in int.cone(\mathcal{P}\cap\mathbb{Z}^d)$ be the vector corresponding to a multiplicity $(d + Z')\gamma$ for a $Z' \leq Z$. Note that by definition of b', we have that $Dist(b') = d + Z'$, as γ is chosen from $(\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$ and the existence of a solution λ'' for b' with smaller vertex distance would imply a small vertex distance for b. Since $b' \in Cone(\mathcal{P} \cap \mathbb{Z}^d)$, there exist a basic feasible solution $\lambda^f \in \mathbb{R}^d_{\geq 0}$ corresponding to vectors $B_1, \ldots, B_d \in \mathcal{P} \cap \mathbb{Z}^d$ of LP (5.5) with $b' = \lambda^f_1 B_1 + \ldots + \lambda^f_d B_d$ and $\left\|\lambda^f\right\|_1 \leq d + Z'$. Using Caratheodory's theorem, we can assume w.l.o.g. that $B_1, \ldots, B_d$ are vertices.

Claim: The vector $[b'] = \{\lambda^f_1\}B_1 + \ldots + \{\lambda^f_d\}B_d$ does not fulfill the integer roundup property.
Suppose the roundup property for $[b']$ is fulfilled, then there exists a packing of instance $[b']$ into $\lceil \left\|\{\lambda^f\}\right\|_1 \rceil$ bins. Using the decomposition of $b' = B\lambda^{f^{int}} + [b']$ into an integral part $B\lambda^{f^{int}} = \lfloor\lambda^f_1\rfloor B_1 + \ldots + \lfloor\lambda^f_d\rfloor B_d$ and the fractional part $[b']$, we obtain a packing for b' into $\lceil \left\|\lambda^f\right\|_1 \rceil \leq d + Z'$ bins (which implies optimality). The constructed packing has vertex distance of $\leq \lceil \left\|\{\lambda^f\}\right\|_1 \rceil$. Since $\lceil \left\|\{\lambda^f\}\right\|_1 \rceil \leq d < d + Z' = Dist(b')$, this is a contradiction to the minimality of the vertex distance for b'.

Claim: Let vectors $b^{(1)}, b^{(2)} \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ be given which correspond to multiplicities $K_1\gamma$ and $K_2\gamma$ of vector $\gamma \in (\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$ with $K_1 < K_2$. Then $[b^{(1)}] \neq [b^{(2)}]$.
By a similar argument as in the previous claim, we can argue in this case. Since $b^{(1)}, b^{(2)} \in Cone(\mathcal{P} \cap \mathbb{Z}^d)$, there exist basic feasible solutions $\lambda^{(1)}, \lambda^{(2)} \in \mathbb{R}^d_{\geq 0}$ corresponding to vectors

$B_1, \ldots, B_d \in V_I$ of LP (5.5) with $b^{(i)} = \lambda_1^{(i)} B_1 + \ldots + \lambda_d^{(i)} B_d$ for $i = 1, 2$ and $\lambda^{(1)} \leq \lambda^{(2)}$ as $K_1\gamma \leq K_2\gamma$. Suppose that $[b^{(1)}] = [b^{(2)}]$, then we obtain for the difference $(K_2 - K_1)\gamma$ corresponding to $b^{(2)} - b^{(1)} = B{\lambda^{(2)}}^{int} + [b^{(2)}] - B{\lambda^{(1)}}^{int} - [b^{(1)}] = B{\lambda^{(2)}}^{int} - B{\lambda^{(1)}}^{int}$. Therefore, the difference $b^{(2)} - b^{(1)}$ can be written by the (positive) sum of vertices $B_1, \ldots, B_d$. This implies a packing for $b^{(2)}$ by $b^{(2)} = b^{(1)} + (b^{(2)} - b^{(1)})$ with vertex distance $Dist(b^{(1)}) < Dist(b^{(2)})$ which contradicts the minimality of vertex distance $b^{(2)}$.

As a conclusion of the above claims, we obtain for each multiplicity $(d+1)\gamma, \ldots, (d+Z)\gamma$ of γ the instances $[(d+1)\gamma], \ldots, [(d+Z)\gamma] \in G(\Pi)$, where die IRUP does not hold. □

Note that an instance with large vertex distance (e.g. double exponential in d) implies the existence of solutions with large (double exponential) multiplicities λ_γ as the number of non-zero components can be bounded by the theorem of Eisenbrand and Shmonin [ES06] applied to points in $(\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$.

Together with the construction of the previous subsection where we created a bin packing instance b with a unique solution $\lambda \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ with $\lambda_\gamma = 2^{2^{\Omega(d)}}$ for some $\gamma \in \mathcal{P} \cap \mathbb{Z}^d$, we obtain that b has double exponentially many subinstances where the IRUP is not fulfilled.

Bibliography

[Alo+97] N. Alon, Y. Azar, G. Woeginger, and T. Yadid. "Approximation Schemes for Scheduling". In: *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '97. 1997, pp. 493–500.

[Alo+98] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. "Approximation schemes for scheduling on parallel machines". In: *Journal of Scheduling* 1 (1998), pp. 55–66.

[Bal+08] J. Balogh, J. Békési, G. Galambos, and G. Reinelt. "Lower Bound for the Online Bin Packing Problem with Restricted Repacking". In: *SIAM Journal on Computing* 38.1 (2008), pp. 398–410.

[Bar07] A. Barvinok. "Lattice points, polyhedra, and complexity". In: *Geometric Combinatorics, IAS/Park City Mathematics Series* 13 (2007), pp. 19–62.

[BB10] A. Beloglazov and R. Buyya. "Energy Efficient Allocation of Virtual Machines in Cloud Data Centers". In: *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid 2010*. 2010, pp. 577–578.

[BBG12] J. Balogh, J. Békési, and G. Galambos. "New lower bounds for certain classes of bin packing algorithms". In: *Theoretical Computer Science* 440-441 (2012), pp. 1–13.

[BJK15] S. Berndt, K. Jansen, and K. Klein. "Fully Dynamic Bin Packing Revisited". In: *18th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. 2015, pp. 135–151.

[BKB07] N. Bobroff, A. Kochut, and K.A. Beaty. "Dynamic Placement of Virtual Machines for Managing SLA Violations". In: *Integrated Network Management, IM 2007. 10th IFIP/IEEE International Symposium on Integrated Network Management*. 2007, pp. 119–128.

[BM98] P.A. Beling and N. Megiddo. "Using Fast Matrix Multiplication to Find Basic Solutions". In: *Theoretical Computer Science* 205.1–2 (1998), pp. 307–316.

[Bro79] D.J. Brown. *A Lower Bound for On-Line One-Dimensional Bin Packing Algorithms*. Tech. rep. R-864. Coordinated Sci Lab Univ of Illinois Urbana, 1979.

[Cap+14] A. Caprara, M. Dell'Amico, José C. Díaz-Díaz, M. Iori, and R. Rizzi. "Friendly bin packing instances without Integer Round-up Property". In: *Mathematical Programming* 150.1 (2014), pp. 5–17.

[CGJ83] E.G. Coffman, M.R. Garey, and D.S. Johnson. "Dynamic Bin Packing". In: *SIAM Journal on Computing* 12.2 (1983), pp. 227–258.

[CGJ84] E.G. Coffman, M.R. Garey, and D.S. Johnson. "Approximation Algorithms for Bin-Packing: An updated Survey". In: *Algorithm design for computer system design* (1984), pp. 49–106.

[CGJ97] E.G. Coffman, M.R. Garey, and D.S. Johnson. "Approximation Algorithms for Bin Packing: A Survey". In: *Approximation algorithms for NP-hard problems*. Ed. by D. Hochbaum. PWS Publishing Co., 1997, pp. 46–93.

[CJZ13] L. Chen, K. Jansen, and G. Zhang. "On the optimality of approximation schemes for the classical scheduling problem". In: *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*. Society for Industrial and Applied Mathematics, 2013, pp. 657–668.

[CLW08] J.W. Chan, T. Lam, and P.W.H. Wong. "Dynamic Bin Packing of Unit Fractions Items". In: *Theoretical Computer Science* 409.3 (2008), pp. 521–529.

[Cof+13] E. G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo. "Bin Packing Approximation Algorithms: Survey and Classification". In: *Handbook of Combinatorial Optimization*. Ed. by M. P. Pardalos, D. Du, and L. R. Graham. Springer New York, 2013, pp. 455–531. ISBN: 978-1-4419-7997-1.

[Coo+86] W. Cook, A.M.H. Gerards, A. Schrijver, and E. Tardos. "Sensitivity theorems in integer linear programming". In: *Mathematical Programming* 34.3 (1986), pp. 251–264.

[Coo+92] W. Cook, M. Hartmann, R. Kannan, and C. McDiarmid. "On integer points in polyhedra". English. In: *Combinatorica* 12.1 (1992), pp. 27–37. DOI: 10.1007/BF01191202.

[CW96] J. Csirik and G.J. Woeginger. "On-line Packing and Covering Problems". In: *Online Algorithms*. Vol. 1442. LNCS. 1996, pp. 147–177.

[CWY09] J.W. Chan, P.W.H. Wong, and F.C.C. Yung. "On Dynamic Bin Packing: An Improved Lower Bound and Resource Augmentation Analysis". In: *Algorithmica* 53.2 (2009), pp. 172–206.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. ISBN: 978-1-4612-6798-0.

[DKL14] K. Daudjee, S. Kamali, and A. López-Ortiz. "On the Online Fault-tolerant Server Consolidation Problem". In: *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '14*. 2014, pp. 12–21.

[Eis57] K. Eisemann. "The Trim Problem". In: *Management Science* 3.3 (1957), pp. 279–284.

[EL09] L. Epstein and A. Levin. "A robust APTAS for the Classical Bin Packing Problem". In: *Mathematical Programming* 119.1 (2009), pp. 33–49.

[EL13] L. Epstein and A. Levin. "Robust Approximation Schemes for Cube Packing". In: *SIAM Journal on Optimization* 23.2 (2013), pp. 1310–1343.

[EL14] L. Epstein and A. Levin. "Robust Algorithms for Preemptive Scheduling". In: *Algorithmica* 69.1 (2014), pp. 26–57.

[ES06] F. Eisenbrand and G. Shmonin. "Carathéodory bounds for integer cones". In: *Operations Research Letters* 34 (2006), pp. 564–568.

[FL81] W. Fernandez de la Vega and G.S. Lueker. "Bin Packing can be solved within $1 + \epsilon$ in Linear Time". In: *Combinatorica* 1.4 (1981), pp. 349–355.

[GIS77] T. F. Gonzalez, O. H. Ibarra, and S. Sahni. “Bounds for LPT Schedules on Uniform Processors”. In: *SIAM Journal on Computing* 6.1 (1977), pp. 155–166.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 2nd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1994. ISBN: 0201558025.

[GPT00] G. Gambosi, A. Postiglione, and M. Talamo. “Algorithms for the Relaxed Online Bin-Packing Model”. In: *SIAM Journal on Computing* 30.5 (2000), pp. 1532–1551.

[GR14] M. X. Goemans and T. Rothvoß. “Polynomiality for bin packing with a constant number of item types”. In: *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA ’14)*. 2014, pp. 830–839.

[Gra66] R. L. Graham. “Bounds for certain multiprocessing anomalies”. In: *Bell System Technical Journal* 45 (1966), pp. 1563–1581.

[Gra69] R. L. Graham. “Bounds on multiprocessing timing anomalies”. In: *SIAM Journal on Applied Mathematics* 17 (1969), pp. 416–429.

[Gri+01] M.D. Grigoriadis, L.G. Khachiyan, L. Porkolab, and J. Villavicencio. “Approximate Max-Min Resource Sharing for Structured Concave Optimization”. In: *SIAM Journal on Optimization* 11 (2001), p. 1081.

[Har89] M. Hartmann. “Cutting planes and the complexity of the integer hull, Report No. 819”. PhD thesis. Cornell University, 1989.

[HL83] A.C. Hayes and D.G. Larman. “The vertices of the knapsack polytope”. In: *Discrete Applied Mathematics* 6.2 (1983), pp. 135–138.

[Hoc97] D. Hochbaum, ed. *Approximation algorithms for NP-hard problems*. PWS Publishing Company, 1997.

[HR15] R. Hoberg and T. Rothvoss. “A Logarithmic Additive Integrality Gap for Bin Packing”. In: *CoRR* abs/1503.08796 (2015).

[HS16] S. Heydrich and R. van Stee. “Beating the Harmonic Lower Bound for Online Bin Packing”. In: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*. 2016, 41:1–41:14.

[HS87] D. S. Hochbaum and D. B. Shmoys. “Using dual approximation algorithms for scheduling problems: theoretical and practical results”. In: *Journal of the ACM* 34 (1987), pp. 144–162.

[IL09] Z. Ivković and E.L. Lloyd. “Fully Dynamic Bin Packing”. In: *Fundamental Problems in Computing*. Springer, 2009, pp. 407–434.

[IL97] Z. Ivković and E.L. Lloyd. “Partially Dynamic Bin Packing can be solved within $1+\epsilon$ in (amortized) Polylogarithmic Time”. In: *Information Processing Letter* 63.1 (1997), pp. 45–50.

[IL98] Z. Ivković and E.L. Lloyd. “Fully Dynamic Algorithms for Bin Packing: Being (Mostly) Myopic Helps”. In: *SIAM Journal on Computing* 28.2 (1998), pp. 574–611.

[Jan06] K. Jansen. “Approximation Algorithms for Min-Max and Max-Min Resource Sharing Problems, and Applications”. In: *Efficient Approximation and Online Algorithms*. Vol. 3484. LNCS. Springer, 2006, pp. 156–202.

[Jan10] K. Jansen. “An EPTAS for Scheduling Jobs on Uniform Processors: Using an MILP Relaxation with a Constant Number of Integral Variables”. In: *SIAM Journal on Discrete Mathematics* 24 (2010), pp. 457–485.

[JK] K. Jansen and K. Klein. “About the Structure of the Integer Cone and its Application to Bin Packing”. In: *Symposium on Discrete Algorithms, SODA (to appear)*.

[JK13] K. Jansen and K. Klein. “A Robust AFPTAS for Online Bin Packing with Polynomial Migration”. In: *International Colloquium on Automata, Languages, and Programming(ICALP)*. 2013, pp. 589–600.

[JKV16] K. Jansen, K. Klein, and J. Verschae. “Closing the Gap for Makespan Scheduling via Sparsification Techniques”. In: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*. 2016, 72:1–72:13.

[JM10] K. Jansen and M. Mastrolilli. “Scheduling unrelated parallel machines: linear programming strikes back”. In: *University of Kiel, Technical Report* 1004 (2010).

[Joh+74a] D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, and R.L. Graham. “Worst-case Performance Bounds for Simple One-dimensional Packing Algorithms”. In: *SIAM Journal on Computing* 3.4 (1974), pp. 299–325.

[Joh+74b] D.S. Johnson, A.J. Demers, J.D. Ullman, M.R. Garey, and R.L. Graham. “Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms”. In: *SIAM Journal on Computing* 3.4 (1974), pp. 299–325.

[Joh74] D.S. Johnson. “Fast Algorithms for Bin Packing”. In: *Journal of Computer and System Sciences* 8.3 (1974), pp. 272–314.

[JR11] K. Jansen and C. Robenek. “Scheduling Jobs on Identical and Uniform Processors Revisited”. In: *Approximation and Online Algorithms*. Ed. by R. Solis-Oba and G. Persiano. Lecture Notes in Computer Science 7164. 2011, pp. 109–122.

[JS11] 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.

[Jun+08] G. Jung, K.R. Joshi, M.A. Hiltunen, R.D. Schlichting, and C. Pu. “Generating Adaptation Policies for Multi-tier Applications in Consolidated Server Environments”. In: *2008 International Conference on Autonomic Computing, ICAC 2008, June 2-6, 2008, Chicago, Illinois, USA*. 2008, pp. 23–32.

[Jun+09] G. Jung, K.R. Joshi, M.A. Hiltunen, R.D. Schlichting, and C. Pu. “A Cost-sensitive Adaptation Engine for Server Consolidation of Multitier Applications”. In: *Middleware 2009, ACM/IFIP/USENIX, 10th International Middleware Conference, Proceedings*. 2009, pp. 163–183.

[Kan87] R. Kannan. “Minkowski’s Convex Body Theorem and Integer Programming”. In: *Mathematics of Operations Research* 12 (1987), pp. 415–440.

[KK82] N. Karmarkar and R.M. Karp. “An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem”. In: *23rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1982, pp. 312–320.

[Len83] H. W. Lenstra. “Integer Programming with a Fixed Number of Variables”. In: *Mathematics of Operations Research* 8 (1983), pp. 538–548.

[Leu89] J. Y-T. Leung. “Bin packing with restricted piece sizes”. In: *Information Processing Letters* 31 (1989), pp. 145–149.

[Lia80] F.M. Liang. “A Lower Bound for On-line Bin Packing”. In: *Information processing letters* 10.2 (1980), pp. 76–79.

[LJ83] H. W. Lenstra and Jr. “Integer programming with a fixed number of variables”. In: *Mathematics of Operations Research* 8.4 (1983), pp. 538–548.

[LL85] C.C. Lee and D. Lee. “A Simple On-line Bin-Packing Algorithm”. In: *Journal of the ACM (JACM)* 32.3 (1985), pp. 562–572.

[LTC14] Y. Li, X. Tang, and W. Cai. “On Dynamic Bin Packing for Resource Allocation in the Cloud”. In: *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA ’14*. 2014, pp. 2–11.

[Onn15] S. Onn. “Unimodular Integer Caratheodory is Fixed Parameter Tractable”. In: *CoRR* abs/1511.03403 (2015).

[Par+00] J.M. Park, Uday R. Savagaonkar, E.K.P. Chong, H.J. Siegel, and S.D. Jones. “Efficient Resource Allocation for QoS Channels in MF-TDMA Satellite Systems”. In: *MILCOM 2000. 21st Century Military Communications Conference Proceedings*. Vol. 2. IEEE. 2000, pp. 645–649.

[Rot13] T. Rothvoss. “Approximating Bin Packing within O(log OPT * Log Log OPT) Bins”. In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science* (2013), pp. 20–29.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., 1986. ISBN: 0-471-90854-1.

[SKZ08] S. Srikantaiah, A. Kansal, and F. Zhao. “Energy Aware Consolidation for Cloud Computing”. In: *Proceedings of the 2008 Conference on Power Aware Computing and Systems*. HotPower’08. San Diego, California, 2008, pp. 10–10.

[SSS09] P. Sanders, N. Sivadasan, and M. Skutella. “Online Scheduling with Bounded Migration”. In: *Mathematics of Operations Research* 34.2 (2009), pp. 481–498.

[ST97] G. Scheithauer and J. Terno. “Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem”. In: *Operations Research Letters* 20.2 (1997), pp. 93–100.

[Sta70] H.M. Stark. *An introduction to number theory*. Markham mathematics series. Markham Pub. Co., 1970.

[Sto13] A.L. Stolyar. “An Infinite Server System with General Packing Constraints”. In: *Operations Research* 61.5 (2013), pp. 1200–1217.

[SV10] M. Skutella and J. Verschae. “A Robust PTAS for Machine Covering and Packing”. In: *European Symposium on Algorithms(ESA)*. Vol. 6346. LNCS. 2010, pp. 36–47.

[SV16] Martin Skutella and José 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.

[SZ13] A.L. Stolyar and Y. Zhong. “A Large-scale Service System with Packing Constraints: Minimizing the Number of Occupied Servers”. In: *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*. ACM. 2013, pp. 41–52.

[Ull71] J.D. Ullman. *The Performance of a Memory Allocation Algorithm*. Technical report. Princeton University, 1971.

[VAN08] A. Verma, P. Ahuja, and A. Neogi. “pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems”. In: *Middleware 2008, ACM/IFIP/USENIX 9th International Middleware Conference, Proceedings*. 2008, pp. 243–264.

[Vli92] A. Vliet. “An Improved Lower Bound for On-Line Bin Packing Algorithms”. In: *Information Processing Letters* 43.5 (1992), pp. 277–284.

[Yao80] A.C. Yao. “New Algorithms for Bin Packing”. In: *Journal of the ACM (JACM)* 27.2 (1980), pp. 207–227.

Erklärung

Diese Abhandlung ist nach Inhalt und Form meine eigene Arbeit. Ausnahmen sind die Beratung und Zusammenarbeit mit meinem Betreuer Prof. Dr. Klaus Jansen. Außerdem basieren chapter 3 und chapter 4 dieser Dissertation auf Publikationen [BJK15] bzw. [JKV16], welche in Zusammenarbeit mit meinem Betreuer und mit Sebastian Berndt bzw. Jose Verschae entstanden sind.

Diese Dissertation wurde weder ganz noch in Teilen an anderer Stelle im Rahmen eines Prüfungsverfahrens vorgelegt. Die Veröffentlichungen und Einreichungen zur Veröffentlichung, auf denen die Dissertation basiert, sind auf Seite 5 dieser Dissertation aufgelistet.

Diese Arbeit ist unter Einhaltung der Regeln guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft entstanden.

Ort, Datum

Unterschrift