MAX A. DEPPERT

ALGORITHMS FOR SCHEDULING PROBLEMS AND INTEGER PROGRAMMING

ALGORITHMS FOR SCHEDULING PROBLEMS AND INTEGER PROGRAMMING

MAX A. DEPPERT

Dissertation

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

eingereicht im November 2022

1. GUTACHTER	Prof. Dr. Klaus Jansen, Christian-Albrechts-Universität zu Kiel
2. GUTACHTER	Prof. Dr. Jian-Jia Chen, Technische Universität Dortmund
3. GUTACHTER	Prof. Dr. Peter Sanders, Karlsruher Institut für Technologie

DATUM DER DISPUTATION 19.04.2023

Max A. Deppert: *Algorithms for Scheduling Problems and Integer Programming*,
Dissertation, © November 2022

The aim of argument or of discussion,
should not be victory, but progress.

— Joseph Joubert

Für meine Eltern

ABSTRACT

In this thesis we give new algorithmic results to the fields of scheduling problems and integer programming. The first part gives approximation results to scheduling problems. The classical makespan minimization problem on identical parallel machines asks for a distribution of n jobs to m machines such that the latest job completion time is minimized. For this setting, which is known to be strongly NP-complete and also is denoted as $P||C_{max}$, we give a new efficient polynomial-time approximation scheme (EPTAS) which improves upon the best previously known approximation result. Furthermore, the novelty about this scheme is that it admits a practical implementation which beats the currently best approximation ratio of $13/11 \approx 1.18$ of the MULTIFIT algorithm. A natural and well-studied extension of $P||C_{max}$ is the partition of the jobs into classes which impose a class-specific setup time on a machine whenever the processing switches to a job of a different class. These problems are called scheduling problems with batch setup times and we present a 2- and 1.5-approximation for each of the three major cases of splittable, non-preemptive, and preemptive scheduling. In the latter and most complicated setting our algorithm improves upon the best previously known approximation ratio of ~ 2. We achieve similar results for the likewise natural variant of many shared resources scheduling (MSRS) where instead of imposing a setup time each class is identified by a unit-sized resource which is occupied by each of its jobs at runtime, i. e. jobs of the same class cannot run in parallel time. For MSRS we present a $1.\bar{6}$-approximation which already improves the best previously known ratio of ~ 2, a 1.5-approximation, and two EPTAS results for a fixed number of machines or resource augmentation.

The second part of this thesis presents exact algorithms to real-time scheduling problems and variants of block-structured integer programming. The scheduling problem of uniprocessor real-time task systems stems from practice but still lacks a theoretical understanding, especially in the presence of task release jitters. We generalize upon existing work and give a new approach to compute worst-case response times for real-time task systems which even admits the first polynomial-time algorithm for harmonic task systems in the presence of task release jitters. In more detail, we prove a duality between Response Time Computation (RTC) and the Mixing Set problem. Therefore, it also admits algorithms to solve Mixing Set. In fact, both problems can be expressed as block-structured integer programs, i. e. RTC as a 4-block integer program and Mixing Set as a 2-stage stochastic integer program. Closely related to both problems are Directed Diophantine Approximation and simultaneous congruences. Especially the latter play an important role in both, theory and practice. However, the common setting of the famous Chinese Remainder Theorem is that each congruence has to have a certain remainder. We relax this setting such that the remainder of each congruence may lie in a given interval. Interestingly, the smallest (or largest) solution to these simultaneous congruences can be computed in polynomial time if the set of divisors is harmonic.

ZUSAMMENFASSUNG

In dieser Thesis werden neue algorithmische Ergebnisse für Scheduling-Probleme und ganzzahlige Programmierung gezeigt. Der erste Teil befasst sich mit Approximationsergebnissen für Scheduling-Probleme. Das klassische Scheduling-Problem der Minimierung des Makespans auf identischen parallelen Maschinen fragt nach einer Verteilung von n Jobs auf m Maschinen, welche die letzte Fertigstellungszeit eines Jobs minimiert. Für dieses Problem, das NP-vollständig ist und auch mit $P||C_{max}$ bezeichnet wird, entwickeln wir ein effizientes Polynomialzeit-Approximationsschema (EPTAS), welches das beste bisher bekannte Approximationsresultat verbessert. Das Besondere an diesem Schema ist, dass es eine Implementierung in der Praxis erlaubt, welche die beste bisher bekannte Approximationsrate von $13/11 \approx 1.18$ schlägt, die der MULTIFIT-Algorithmus garantiert. Eine natürliche und populäre Erweiterung des Problems ist die Partitionierung der Jobs in Klassen, welche auf einer Maschine eine klassenspezifische Setup-Zeit erzwingen, sobald ein Job einer anderen Klasse ausgeführt werden soll. Diese Probleme heißen Scheduling-Probleme mit Batch-Setup-Zeiten und wir zeigen eine 2- und 1.5-Approximation für die drei wichtigsten Varianten des Splittable Schedulings, des nicht-präemptiven sowie des präemptiven Schedulings. Für die letzte und schwierigste Variante verbessern wir damit die beste bisher bekannte Approximationsrate von ~ 2. Ähnliche Resultate erzielen wir für das ebenso natürliche Scheduling-Problem mit vielen geteilten Ressourcen (MSRS). Anstatt Setup-Zeiten zu erzwingen, werden die Klassen hier durch eine Einheitsressource identifiziert, die von ihren Jobs zur Laufzeit belegt wird, d.h. Jobs der selben Klasse können nicht parallel ausgeführt werden. Für MSRS präsentieren wir eine $1.\bar{6}$-Approximation (welche bereits die beste bisher bekannte Rate von ~ 2 verbessert), eine 1.5-Approximation sowie zwei EPTAS-Schemata bei konstant vielen Maschinen oder zugelassener Ressourcenaugmentierung.

Im zweiten Teil werden exakte Algorithmen für Real-Time-Scheduling-Probleme und Varianten blockstrukturierter ganzzahliger Programme vorgestellt. Real-Time-Task-Systeme auf einem einzelnen Prozessor stammen aus der Praxis, aber noch immer fehlt ein theoretisches Verständnis insbesondere unter Beachtung von Task-Release-Verzögerungen. Wir verallgemeinern bekannte Arbeiten und zeigen einen neuen Ansatz, um Worst-Case-Response-Zeiten für Real-Time-Task-Systeme zu berechnen, der zudem den ersten Polynomialzeitalgorithmus für harmonische Task-Systeme möglich macht. Genauer gesagt beweisen wir eine Dualität zwischen Response Time Computation (RTC) und Mixing Set. Daher erhalten wir auch Algorithmen, um Mixing Set zu lösen. Tatsächlich können beide Probleme als blockstrukturierte ganzzahlige Programme (IP) ausgedrückt werden, d.h. RTC kann als 4-block IP und Mixing Set als 2-stage-stochastic IP ausgedrückt werden. Mit diesen beiden Problemen eng verwandt sind Directed Diophantine Approximation und simultane Kongruenzen. Besonders letztere spielen sowohl in der Theorie als auch in der Praxis eine wichtige Rolle. Allerdings ist es

der klassischen Situation des chinesischen Restsatzes zu eigen, dass jede Kongruenz einen bestimmten Restwert besitzen muss. Wir verallgemeinern dies und fordern lediglich, dass der Restwert einer Kongruenz innerhalb eines gegebenen Intervalls liegt. Interessanterweise lässt sich die kleinste (oder größte) Lösung solcher Kongruenzen in Polynomialzeit berechnen, sofern die Teiler harmonisch sind.

PUBLICATIONS

The results presented in this thesis are based on the following peer-reviewed publications:

[1] Sebastian Berndt, Max A. Deppert, Klaus Jansen, and Lars Rohwedder. Load balancing: The long road from theory to practice. In *Proc. ALENEX 2022*, pages 104–116. SIAM, 2022. Implementation of the BDJR algorithm: github.com/made4this/BDJR.

[2] Max A. Deppert and Klaus Jansen. Near-linear approximation algorithms for scheduling problems with batch setup times. In *Proc. SPAA 2019*, pages 155–164. ACM, 2019.

[3] Max A. Deppert, Klaus Jansen, and Kim-Manuel Klein. Fuzzy simultaneous congruences. In *Proc. MFCS 2021*, volume 202 of *LIPIcs*, pages 39:1–39:16, 2021.

Furthermore, the following work is publicly available and intended to be published in a peer-reviewed conference or journal:

[4] Max A. Deppert and Klaus Jansen. The power of duality: Response time analysis meets integer programming. *CoRR*, abs/2210.02361, 2022. In submission.

[5] Max A. Deppert, Klaus Jansen, Marten Maack, Simon Pukrop, and Malin Rau. Scheduling with many shared resources. *CoRR*, abs/2210.01523, 2022. To be published in Proc. IPDPS 2023.

I never teach my pupils.
I only attempt to provide the conditions in which they can learn.

— Albert Einstein

ACKNOWLEDGMENTS

First of all I want to thank my advisor Klaus Jansen for all of his answers and even more for all of his questions. His support and introduction to a wealth of optimization problems were of great benefit.

I want to thank Thomas Wilke and Matthias Mnich for hosting me during the last months of my studies. Special thanks goes to Werner Grass for reviewing the real-time scheduling related part of my thesis to provide various helpful comments and suggestions.

Furthermore, among my coworkers, coauthors, and friends, I am grateful for the inspiring time with Sebastian Berndt, Hauke Brinkop, Laura Codazzi, Julia Ehrenbrecht, David Fischer, Katrin Flöth, Esther Galby, Julian Golak, Kilian Grage, Ute Iaquinto, Kai Kahler, Matthias Kaul, Kim-Manuel Klein, Leonie Krull, Alexandra Lassota, Marten Maack, Parvaneh Massouleh, Simon Pukrop, Malin Rau, Janina Reuter, Lars Rohwedder, Henrik Schmidt, Christoph Daniel Schulze, Malte Skambath, Tobias Stamm, and José Verschae. It really was a pleasure to work and be with you and I have always enjoyed your company.

Although the COVID-19 pandemic was not helpful at all, during the time of my PhD studies I visited Hamburg, Bremen, Phoenix, Bergen, Paderborn, Tallinn, Santiago de Chile, Oropa, and Amsterdam. I feel lucky to have had the opportunity to visit all these great places.

Last but not least I want to thank my parents Ulrike and Wolfgang and especially my brother Konrad and my girlfriend Isabel for supporting me in any way and not getting tired of my weirdest research ideas.

CONTENTS

1 INTRODUCTION

The search for efficient algorithms can be traced to the ancient Babylonian mathematicians c. 2500 BCE. Their idea of efficiency must have been the sheer ability to "run" an algorithm on the human brain. Today, c. 4500 years later, we rather consider algorithms to be efficient if their running time, e. g. the total number of arithmetic operations, can be expressed as a function which is polynomial in the encoding size of the input. However, in the early 1970s the unsatisfying complexity conjecture "P ≠ NP" [32, 93] came up, saying that there may be a lot of optimization problems which do not admit algorithms to solve them in polynomial time. Since then, it is also known as the most important open problem of computer science in general. As a member of the list of the seven *Millennium Prize Problems* its solution is worth a million dollars [66] and despite lots of intense tries it could not be solved until today.

Briefly explained, the question is whether it is as simple to *find a solution* to a decision problem as to *verify a given solution* to that problem. The majority of scientists believes that this is not the case, which can be denoted by P ≠ NP where P is the set of problems which can be solved in polynomial time and NP is the set of problems which admit a polynomial-time verification algorithm for potential solutions. However, it is unclear whether P = NP holds true which would have far-reaching consequences, especially to the security of encryption in cryptography. Until an answer is found, we can only rely on the fact that the hardest problems in NP cannot be solved in polynomial time, if P ≠ NP holds. These problems are known as NP-*complete* problems.

Fortunately, we also know that for a lot of these NP-*hard* optimization problems we can design algorithms to find *approximately optimal* solutions in polynomial time. Therefore, in combinatorial optimization there are three main avenues of research. Given an optimization problem which is *hard* to solve under some complexity conjecture, we may either search for more efficient (yet exponential-time) *exact algorithms* which compute optimal solutions, design better *approximation algorithms* with a shorter (polynomial) running time or an improved accuracy, or find stronger reduction proofs to (different) complexity conjectures. In this thesis, our main focus will be on the former two avenues. We will start with some notation and continue with fundamental principles and examples for approximation algorithms. See Section 1.1 for an overview over the thesis.

NOTATION For the sake of readability we introduce common notation. We will always write $\log = \log_2$ to denote the *binary* logarithm. Also we will use $[n] = \{ i \in \mathbb{Z} \mid 1 \leq i \leq n \}$ to refer to the first n natural numbers and the empty set is denoted by $\emptyset$. For any instance I of a given optimization problem P we refer to the encoding size of I by $|I|$ and we denote the objective value of optimum solutions to I by $\mathrm{OPT}_P(I)$. If clear from the context, we also write $\mathrm{OPT}(I)$ or simply OPT.

INTEGER PROGRAMMING A plethora of optimization problems can be expressed with the powerful notion of integer programming. The general integer linear program (ILP) in standard form is the program to compute $\max\{c^\intercal x \mid Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$, given integers $m, n \in \mathbb{Z}_{\geq 1}$, a constraint matrix $A \in \mathbb{Z}^{m \times n}$, a right-hand side $b \in \mathbb{Z}^m$, and an objective vector $c \in \mathbb{Z}^n$. By $\Delta = \|A\|_\infty$ we denote the largest absolute value of the entries in A. The importance of an efficient algorithm to solve this NP-hard problem can not be overestimated, as it is a tool to solve *any* problem that can be stated as an ILP.

Lenstra Jr. [81] showed that an ILP in inequality form with encoding length $|I|$ can be solved in time $2^{\mathcal{O}(n^2)} \cdot |I|^{\mathcal{O}(1)}$, which is polynomial if n is fixed. The former factor was improved to $2^{\mathcal{O}(n \log n)}$ by Kannan [82] and until today it is open whether it can be improved to $2^{\mathcal{O}(n)}$. By complementing the preceding, in 1981 Papadimitriou [106] found an algorithm running in time $(m \cdot \max\{\Delta, \|b\|_\infty\})^{\mathcal{O}(m^2)}$, which is pseudo-polynomial if m is fixed.

After these famous algorithms were presented, the most upcoming results improved constant factors in the running times, and it seemed as if the substantial progress had lapsed, but in 2017 Eisenbrand and Weismantel [47] proved that the ILP in standard form can be solved in time $n \cdot (m\Delta)^{\mathcal{O}(m)} \cdot \|b\|^2_\infty$. Their key argument was the *Steinitz Lemma* [112]. Briefly explained, it implies that a point Ax only has to be considered as a partial solution of b, if it is close to the line $\overline{0b} = \{\lambda \cdot b \mid \lambda \in [0, 1]\}$. Therefore, all potential partial solutions can be enumerated and may be understood as an edge-weighted graph such that the optimal solution can be computed by solving an instance of the LONGEST PATH problem. Unwilling to stop there, this great result was then improved by Jansen and Rohwedder [76] who presented an algorithm running in time $\mathcal{O}(\sqrt{m}\Delta)^{2m} \cdot \log(\|b\|_\infty) + \mathcal{O}(mn)$ to solve the general ILP. Again, roughly explained, the crucial improvement was the idea, that there are only $\log(\|b\|_\infty)$ many points of the line $\overline{0b}$ whose environments have to be considered as potential partial solutions.

N-FOLD INTEGER PROGRAMMING The *block-structured* ILPs have received major attention in the recent past. The most popular among them are the so-called N-fold ILPs. A (generalized) N-fold ILP is an ILP of the form

$$\min\{c^T x \mid \mathcal{A}x = b, \ell \leq x \leq u, x \in \mathbb{Z}^{Nt}\}$$

where we are given natural numbers $N, r, s, t \in \mathbb{Z}_{\geq 1}$, block matrices $A^{(i)} \in \mathbb{Z}^{r \times t}$ and $B^{(i)} \in \mathbb{Z}^{s \times t}$ for each $i \in [N]$, vectors $c, \ell, u \in \mathbb{Z}^{Nt}$, $b \in \mathbb{Z}^{r+Ns}$, and the constraint matrix

$$\mathcal{A} = \begin{pmatrix} A^{(1)} & \dots & A^{(N)} \\ B^{(1)} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & B^{(N)} \end{pmatrix}.$$

The N-fold ILPs and variants thereof have been intensively studied in the literature which contains a series of improved algorithms presented in recent years. For an overview about the majority of these developments, we refer to

the extensive work of Eisenbrand et al. [44]. A state-of-the-art result is as follows.

Theorem 1 (Cslovjecsek et al. [34]). *The N-fold integer programming problem can be solved in time $2^{\mathcal{O}(rs^2)}(rs\Delta)^{\mathcal{O}(r^2s+s^2)}(Nt)^{1+o(1)}$ where Δ is the maximum absolute value of the entries in $\mathcal{A}$.*

We call N the number of blocks, r and s the number of global and local constraints, respectively, and t the number of block variables.

2-STAGE STOCHASTIC INTEGER PROGRAMMING The transposed situation to N-fold ILPs is given by *2-stage stochastic* ILPs [63, 85] which unfortunately are much harder to solve. Here, the matrix $\mathcal{A}$ of a 2-stage integer program is constructed by block matrices $A^{(1)},\dots,A^{(N)} \in \mathbb{Z}^{r\times s}$ and $B^{(1)},\dots,B^{(N)} \in \mathbb{Z}^{r\times t}$ as follows:

$$\mathcal{A} = \begin{pmatrix} A^{(1)} & B^{(1)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{(N)} & 0 & \dots & B^{(N)} \end{pmatrix}.$$

For an objective vector $c \in \mathbb{Z}^{s+Nt}_{\geq 0}$, a right-hand side $b \in \mathbb{Z}^{Nr}$, and bounds $\ell, u \in \mathbb{Z}^{s+Nt}_{\geq 0}$ the 2-stage integer program is formulated as

$$\max\{c^T x \mid \mathcal{A}x = b,\ \ell \leq x \leq u,\ x \in \mathbb{Z}^{s+Nt}\}.$$

By using parallel computation a state-of-the-art algorithm for 2-stage stochastic integer programming yields the following theorem.

Theorem 2 (Cslovjecsek et al. [35]). *The 2-stage stochastic integer programming problem can be solved in time $2^{(2\Delta)^{\mathcal{O}(r(r+s))}} \cdot N \log^{\mathcal{O}(rs)} N$ where Δ is the maximum absolute value of the entries in $\mathcal{A}$.*

MAKESPAN MINIMIZATION ($P||C_{max}$) A fundamental problem in combinatorial optimization is the classical scheduling problem of makespan minimization on identical parallel machines (often denoted[1] by $P||C_{max}$) which asks for a distribution of a set J of $n = |J|$ jobs to $m < n$ machines. Each job $j \in J$ has a processing time p_j and the objective is to minimize the makespan, i. e. the latest completion time of a job. More formally, a *schedule* $\sigma : J \to [m]$ assigns jobs to machines. The load of machine i in schedule σ is $\sum_{j\in\sigma^{-1}(i)} p_j$ and the *makespan* of σ is $\mu(\sigma) = \max_{i\in[m]} \sum_{j\in\sigma^{-1}(i)} p_j$. The goal is to find a schedule σ minimizing $\mu(\sigma)$. See Fig. 1.1 for an example. In this thesis we give approximation algorithms to multiple variants of the problem. See Section 1.1 for an overview.

APPROXIMATION ALGORITHMS NP-hard optimization problems cannot be solved in polynomial time unless P = NP. However, for a variety of problems, this disillusioning fact only restrains the computation of *optimal*

1 This is the *three-field problem classification* $\alpha|\beta|\gamma$ introduced by Graham et al. [54]

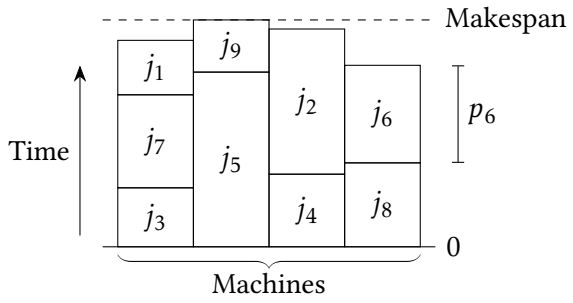


Figure 1.1: A schedule for $n = 9$ jobs $J = \{j_1, \ldots, j_9\}$ on $m = 4$ machines

solutions. Therefore, the computation of *nearly* optimal solutions may be possible in polynomial time. This motivates the investigation of approximation algorithms.

Definition 1 (α-Approximation)**.** *An algorithm $\mathcal{A}$ is an α-approximation for an optimization problem P if for each instance I of P algorithm $\mathcal{A}$ computes a feasible solution to I and*

- $\mathcal{A}(I) \leq \alpha \cdot \mathrm{OPT}_P(I)$. *($P$ is a minimization problem and $\alpha > 1$)*
- $\mathcal{A}(I) \geq \alpha \cdot \mathrm{OPT}_P(I)$. *($P$ is a maximization problem and $\alpha < 1$)*

Clearly, here $\mathcal{A}(I)$ denotes the *objective value* of the solution for instance I computed by algorithm $\mathcal{A}$.

Furthermore, if α is a constant, then an α-approximation is also known as a *constant approximation*. The following example gives a constant approximation to the classical scheduling problem $\mathrm{P}||\mathrm{C}_{\max}$.

Example 1 (2-Approximation for $\mathrm{P}||\mathrm{C}_{\max}$)**.** *To get a 2-approximation for $\mathrm{P}||\mathrm{C}_{\max}$ set $T = \max\{p_{\max}, \frac{1}{m}\sum_{j\in J} p_j\}$ where $p_{\max} = \max_{j\in J} p_j$ is the largest processing time of the instance. Clearly, T is a lower bound for the optimum makespan, i. e. $T \leq \mathrm{OPT}$.*

We place the jobs in an arbitrary order in a greedy fashion as follows. Starting on the first machine we place jobs until a job crosses the border T. We place the job anyway and turn to the next machine and place jobs until the border T is crossed and so on.

In this way, every machine is filled with a load of at least T, which implies that all jobs can be placed, and per machine there is at most one job which uses processing time after time T. Hence, the resulting schedule has a makespan of at most $T + p_{\max} \leq 2 \cdot T \leq 2 \cdot \mathrm{OPT}$ and it can be computed in time $\mathcal{O}(n)$.

APPROXIMATION SCHEMES Unless there is a corresponding result of *inapproximability*, constant approximation algorithms always lead to the natural question "Can we do better?" and in fact, many optimization problems can be approximated to *any* precision. This is the subject of approximation schemes.

Definition 2 (Approximation Scheme)**.** *An approximation scheme for an optimization problem P is a family of algorithms $(\mathcal{A}_\varepsilon)_{\varepsilon>0}$ such that for a given approximation accuracy parameter $\varepsilon > 0$ and an instance I of P, algorithm $\mathcal{A}_\varepsilon$*

QPTAS	PTAS	EPTAS	FPTAS
$\lvert I\rvert^{f(1/\varepsilon)\log^{\mathcal{O}(1)}(\lvert I\rvert)}$	$\lvert I\rvert^{f(1/\varepsilon)}$	$f(1/\varepsilon)\cdot\lvert I\rvert^{\mathcal{O}(1)}$	$(\lvert I\rvert/\varepsilon)^{\mathcal{O}(1)}$

Table 1.1: Running times for common approximation schemes

computes a feasible solution to I whose objective value differs from $\mathrm{OPT}_P(I)$ *by at most* $\varepsilon\cdot\mathrm{OPT}_P(I)$*, i. e.* $|\mathcal{A}_\varepsilon(I)-\mathrm{OPT}_P(I)|\leq\varepsilon\,\mathrm{OPT}_P(I)$.

Clearly, here $\mathcal{A}_\varepsilon(I)$ denotes the *objective value* of the solution for instance I computed by algorithm $\mathcal{A}_\varepsilon$. The parameter ε controls the distance to the optimal objective value, i. e. smaller values of ε result in better solutions. As such, approximation schemes should not be confused with simple *heuristic* algorithms, which may not provide certain approximation guarantees.

Depending on the running time, an approximation scheme is called a quasipolynomial-time approximation scheme (QPTAS) if their algorithms run in time $|I|^{f(1/\varepsilon)\log^{\mathcal{O}(1)}(|I|)}$, a polynomial-time approximation scheme (PTAS) if they run in time $|I|^{f(1/\varepsilon)}$, an efficient polynomial-time approximation scheme (EPTAS) if they run in time $f(1/\varepsilon)\cdot|I|^{\mathcal{O}(1)}$, and a fully polynomial-time approximation scheme (FPTAS) if they run in time $(|I|/\varepsilon)^{\mathcal{O}(1)}$ which is polynomial in $|I|$ as well as in $1/\varepsilon$ (see also Table 1.1).

Observation 1. *Any family of approximation algorithms $(\mathcal{A}_\varepsilon)_{\varepsilon>0}$ for an optimization problem P, with $|\mathcal{A}_\varepsilon(I)-\mathrm{OPT}_P(I)|\leq\mathcal{O}(\varepsilon)\,\mathrm{OPT}_P(I)$ for all $\varepsilon>0$ and all instances I of P, can be turned into an approximation scheme.*

Proof. Given such a family of algorithms there is a constant $c>0$ such that $|\mathcal{A}_\varepsilon(I)-\mathrm{OPT}_P(I)|\leq c\varepsilon\,\mathrm{OPT}_P(I)$ for all $\varepsilon>0$ and all instances I of P. Hence, we define a new family of algorithms $(\mathcal{B}_\varepsilon)_{\varepsilon>0}$ by setting $\mathcal{B}_\varepsilon=\mathcal{A}_{\varepsilon/c}$ for all $\varepsilon>0$. Then, for all $\varepsilon>0$ and all instances I of P it holds that

$$|\mathcal{B}_\varepsilon(I)-\mathrm{OPT}_P(I)|=|\mathcal{A}_{\varepsilon/c}(I)-\mathrm{OPT}_P(I)|\leq\varepsilon\,\mathrm{OPT}_P(I)$$

and thus, we arrive at the desired approximation accuracy of an approximation scheme, howbeit probably slower than before. □

DUAL APPROXIMATION The well-known idea of *dual approximation* was introduced by Hochbaum and Shmoys [64] in 1987. Originally, it was presented as a technique to approximate scheduling problems. However, today it is an omnipresent approach to approximate optimization problems of any kind. Therefore, we will give a more general description as follows.

Consider an optimization problem P and a constant $\rho>0$. Then we call an algorithm $\mathcal{A}$ a *ρ-dual approximation algorithm* for P, if it has the following property.

Given a pair (I,g) of an instance I and a guess g of an optimum solution value, $\mathcal{A}$ either *accepts* the input, i. e. it computes a solution to instance I whose objective value is ρg or better, or $\mathcal{A}$ *rejects* the input which then implies that there is no feasible solution for I with objective value equal to or better than g.

The power of such an algorithm is revealed in the presence of bounds for the optimum objective value and an approximation parameter $\varepsilon>0$. If we

know an interval $[\ell, u]$ such that $u \leq \mathcal{O}(\ell)$ and $\mathrm{OPT}_P(I) \in [\ell, u]$, then by using a binary search running in time $\mathcal{O}(f(|I|)\log(\rho/\varepsilon))$ we can compute a solution whose objective value is at most $\varepsilon\,\mathrm{OPT}_P(I)$ worse than $\rho\,\mathrm{OPT}_P(I)$ where $f(|I|)$ is an upper bound on the running time of the ρ-dual approximation algorithm. Thus, for a minimization problem this yields a $(\rho+\varepsilon)$-approximation and for a maximization problem it gives a $(\rho-\varepsilon)$-approximation.

Obviously, a 1-dual (or $(1 \pm \varepsilon)$-dual) approximation algorithm reveals an approximation scheme and for the rest of the section we will use $\mathrm{P}\|\mathrm{C}_{\max}$ as an example to motivate general construction ideas for approximation schemes using the dual approximation approach.

SMALL ITEMS A powerful and at first sight counter-intuitive[2] fact about the approach of dual approximation is that usually, the placement of *small* items can be postponed to the very last step by doing negligible harm to the quality of the solution.

Consider a makespan guess $T \geq \frac{1}{m}\sum_{j\in J} p_j$ (cf. Example 1) and divide the jobs J of a scheduling instance I into *small* jobs J_{small} with $p_j \leq \varepsilon T$ for all $j \in J_{\text{small}}$ and *big* jobs J_{big} with $p_j > \varepsilon T$ for all $j \in J_{\text{big}}$.

Furthermore, suppose that we are given an algorithm to compute a feasible schedule σ_{big} for the big jobs J_{big} with makespan $T \leq (1+\varepsilon)\mathrm{OPT}(I)$. Let $\ell_1, \ldots, \ell_m$ denote the loads of the machines in the new schedule σ obtained by greedily adding the small jobs J_{small} to schedule σ_{big}. We do a simple case distinction as follows. If the makespan has not increased, then σ has a makespan of $\ell_{\max} \leq T \leq (1+\varepsilon)\mathrm{OPT}(I)$. Otherwise, if the makespan is increased, we have $\ell_{\max} - \ell_i \leq \varepsilon T$ for all $i \in [m]$ and thus, we obtain

$$\mathrm{OPT}(I) \geq \frac{1}{m}\sum_{j\in J} p_j = \frac{1}{m}\sum_{i\in[m]} \ell_i \geq \frac{1}{m}\sum_{i\in[m]} (\ell_{\max} - \varepsilon T) = \ell_{\max} - \varepsilon T$$

and we conclude that σ has a makespan of

$$\ell_{\max} \leq \mathrm{OPT}(I) + \varepsilon T \leq (1+\varepsilon(1+\varepsilon))\mathrm{OPT}(I) \leq (1+2\varepsilon)\mathrm{OPT}(I).$$

As observed in Observation 1 this leads to a proper approximation scheme and therefore, small jobs often are not considered at all. In the following we only care about big jobs with $p_j > \varepsilon T$.

SCALING AND ROUNDING Mainly for the sake of simplicity it is common to scale all processing times by $1/T$. As $T \geq p_{\max}$ this especially implies that the scaled processing times are bounded by 1.

To reduce the number of different processing times it is common to use an appropriate *rounding scheme*. While simplifying the instance on the one hand, a proper rounding scheme also has to guarantee that the resulting lack of accuracy is small enough to admit an approximation scheme on the other hand. In the following we will cover the two well-known approaches of *arithmetic rounding* and *geometric rounding*.

The idea of *arithmetic rounding* is to round each processing time p_j down to the largest integer multiple of ε^2 (which is at most p_j), i. e. $p'_j = \lfloor p_j/\varepsilon^2 \rfloor \varepsilon^2$.

2 "What, if there are *a lot* of small jobs?"

Hence, $p'_j \leq p_j < p'_j + \varepsilon^2$ and clearly, the total number of different rounded processing time is $\lfloor p_{\max}/\varepsilon^2 \rfloor \leq \lfloor 1/\varepsilon^2 \rfloor \leq \mathcal{O}(1/\varepsilon^2)$. To see that this simplification admits an approximation scheme, suppose that we have a set of jobs $B \subseteq J$ whose *rounded jobs* are to be scheduled on a single machine, i. e. $\sum_{j \in B} p'_j \leq 1$. Replacing each rounded job by its original job only leads to a small inaccuracy, since

$$\sum_{j \in B} p_j \leq \sum_{j \in B} (p'_j + \varepsilon^2) = \underbrace{\sum_{j \in B} p'_j}_{\leq 1} + \underbrace{|B|}_{\leq 1/\varepsilon} \varepsilon^2 \leq 1 + \varepsilon.$$

If ε is small, e. g. $\varepsilon \leq 1$, instead of arithmetic rounding we can also use *geometric rounding* to reduce the number of different rounded processing times even more. By rounding processing time p_j down to $p'_j = (1+\varepsilon)^{\lfloor \log_{1+\varepsilon}(p_j) \rfloor}$, which simply is the largest power of the sum $1 + \varepsilon$ that does not exceed p_j, we find that the total number of different rounded processing times is at most

$$\begin{aligned} s &= \lfloor \log_{1+\varepsilon}(p_{\max}) \rfloor - \lfloor \log_{1+\varepsilon}(p_{\min}) \rfloor + 1 \\ &\leq \lfloor \log_{1+\varepsilon}(1) \rfloor - \lfloor \log_{1+\varepsilon}(\varepsilon) \rfloor + 1 \\ &\leq \mathcal{O}(\log_{1+\varepsilon}(1/\varepsilon)) \leq \mathcal{O}(\tfrac{1}{\log(1+\varepsilon)} \log(1/\varepsilon)) \leq \mathcal{O}(1/\varepsilon \log(1/\varepsilon)) \end{aligned}$$

where we use $\log(1+\varepsilon) \geq \varepsilon/2$ which follows from $1 + \varepsilon \geq 2^{\varepsilon/2}$ and $\varepsilon \leq 1$. Again, let $B \subseteq J$ be a set of jobs whose rounded jobs are supposed to be scheduled on a single machine. Hence, we have $\sum_{j \in B} p'_j \leq 1$ and because of $p_j \leq (1+\varepsilon) p'_j$ it follows that

$$\sum_{j \in B} p_j \leq \sum_{j \in B} (1+\varepsilon) p'_j = (1+\varepsilon) \sum_{j \in B} p'_j \leq 1 + \varepsilon.$$

In the seminal work [64] of Hochbaum and Shmoys, a PTAS for $P\|C_{\max}$ is proven to run in time $(n/\varepsilon)^{\mathcal{O}(1/\varepsilon^2)} \leq n^{\mathcal{O}(1/\varepsilon^2 \log(1/\varepsilon))}$ by using arithmetic rounding. In the following example we adapt their approach and use geometric rounding to improve the running time to $(n/\varepsilon)^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon))} \leq n^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))}$.

Example 2 (PTAS for $P\|C_{\max}$)**.** *A PTAS for $P\|C_{\max}$ can be constructed as follows. Consider an approximation parameter $\varepsilon > 0$. If $\varepsilon \geq 1$, then it holds that $1 + \varepsilon \geq 2$ and we can use the 2-approximation of Example 1 to compute a sufficient schedule.*

From now on suppose that $0 < \varepsilon < 1$. At first we use the 2-approximation of Example 1 to compute an upper bound u of the optimum makespan and set $\ell = \max\{p_{\max}, \frac{1}{m} \sum_{j \in J} p_j\}$ as a lower bound of the optimum makespan, i. e. $\ell \leq \mathrm{OPT} \leq u \leq 2 \cdot \mathrm{OPT}$. Aiming for a binary search approach let $T \in [\ell, u]$ be a guess of the optimum makespan and scale all processing times by $1/T$. We discard small jobs with processing time at most ε and add them in the end as discussed before. For the large jobs j with processing time $p_j > \varepsilon$ we do geometric rounding and as described above, this results in a total number of $t \leq \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ many different rounded processing times $s_1, \ldots, s_t$ with multiplicities $a_1, \ldots, a_t \geq 1$ (with $\sum_{i \leq t} a_i = n$).

Now, the crucial step is to solve a dynamic program for a suitable instance of Bin Packing. *If $B(b_1,\dots,b_t)$ denotes the minimum total number of bins to store b_i many items of size s_i for each $i \in [t]$, the defining equality is*

$$B(b_1,\dots,b_t) = 1 + \min_{\substack{c\in\mathbb{Z}^t_{\geq 0}\\ c\leq b\\ c^\intercal s\leq 1}} B(b_1-c_1,\dots,b_t-c_t)$$

and we are interested in the value $m_{\min} = B(a_1,\dots,a_t)$ which is exactly the minimum number of bins (or machines) to store (or schedule) our rounded instance.

Obviously, the table B has $\mathcal{O}(n^t)$ many entries and each entry can be computed in time $\mathcal{O}((1/\varepsilon)^t)$. This gives a running time of $\mathcal{O}((n/\varepsilon)^t)$ to compute $m_{\min}$.

Now, go on in the binary search depending on $m_{\min} \leq m$ (which means that T admits a feasible schedule) or $m_{\min} > m$ (which means that $T < \mathrm{OPT}$). This results in a total running time of

$$\begin{aligned}\mathcal{O}(n) + \mathcal{O}(\log(1/\varepsilon)(n/\varepsilon)^t) &\leq \log(1/\varepsilon)(n/\varepsilon)^{\mathcal{O}(1/\varepsilon\log(1/\varepsilon))}\\ &\leq n^{\mathcal{O}(1/\varepsilon\log^2(1/\varepsilon))}.\end{aligned}$$

In fact, an EPTAS for $P||C_{\max}$ can be constructed by using the same approach of Example 2 except for the computation of $m_{\min}$ which may also be computed by solving the so-called *configuration* ILP using the famous algorithms of Lenstra or Kannan. We briefly describe the approach in the following example.

Example 3 (EPTAS for $P||C_{\max}$)**.** *Here we refine the approach of Example 2. If $\mathcal{C} = \{c \in \mathbb{Z}^t_{\geq 0} \mid c \leq a,\ c^\intercal s \leq 1\}$ denotes the set of feasible configurations, then $m_{\min}$ may also be computed as*

$$m_{\min} = \min\{\|x\|_1 : \sum_{c\in\mathcal{C}} x_c c = a,\ x \in \mathbb{Z}^{\mathcal{C}}_{\geq 0}\}.$$

Now, the computation of $m_{\min}$ can easily be understood as an ILP I' where the columns of the matrix are the feasible configurations, i. e. the elements of $\mathcal{C}$. Hence, the corresponding number of variables N is the number of feasible configurations, i. e. $N = |\mathcal{C}| \leq (1/\varepsilon)^t \leq (1/\varepsilon)^{\mathcal{O}(1/\varepsilon\log(1/\varepsilon))}$. Therefore, by directly applying the algorithm of Lenstra [81] or Kannan [82] with a running time of at most $2^{\mathcal{O}(N^2)}\cdot|I'|^{\mathcal{O}(1)}$ to $M = t \leq \mathcal{O}(1/\varepsilon\log(1/\varepsilon))$ constraints whose entries are all bounded by n, this approach yields an EPTAS.

A natural question is to ask for an FPTAS for $P||C_{\max}$. In fact, Horowitz and Sahni [65] give an FPTAS for the problem if the number of machines m is fixed. However, in general the problem is strongly NP-hard which can be proven by a reduction from 3-PARTITION [51]. Therefore, there is no FPTAS for $P||C_{\max}$ unless P = NP.

1.1 OVERVIEW

Here we give a brief overview over the chapters in this thesis.

1.1.1 *Approximation Algorithms*

MAKESPAN MINIMIZATION For $P\|C_{max}$ we obtain a PTAS implementation that achieves in reasonable time a precision which beats the best known guarantee of a polynomial-time non-PTAS algorithm. This precision is $2/11 \approx 18.2\%$, which is guaranteed by the MULTIFIT algorithm. The algorithm we use, which is based on the JR-algorithm, reduces the problem to performing $\mathcal{O}(\log(n))$ many fast Fourier transformations (FFTs), where the size of the FFT input depends only on ε and not the instance itself. Hence, the running time for all instances (using the same precision) is very stable and predictable. This is in the spirit of an EPTAS running time. *Chapter 2*

We successfully run experiments of our implementation for a precision of $\varepsilon < 2/11$ and thus make the claim that this precision is practically feasible in general. This is also the main message of our result. For completeness, we provide comparisons of the solution quality obtained empirically. While the theoretical guarantee of the PTAS is better, the difference to non-PTAS algorithms is marginal at this state and it is not yet evident in the experiments. The execution of the PTAS is computationally expensive and the considered precision is on the edge of what is realistic for our implementation. Nevertheless, the successful execution with a low precision value forms a proof of concept for practical PTAS's.

SCHEDULING WITH BATCH SETUP TIMES The well-studied field of scheduling problems with setup times originates in the natural idea that a job may impose a preparation of its processing machine. A general formulation divides the jobs into classes whose jobs can be processed by a machine consecutively after waiting for an initial single setup time which depends on the associated class. Job sequences scheduled in this manner are called batches. We aim to find a schedule which minimizes the makespan. *Chapter 3*

In this chapter we consider the three most common cases of scheduling problems with batch setup times which are the non-preemptive case where jobs cannot be preempted, the preemptive case where jobs may be preempted but not processed in parallel time, and the splittable case where jobs may be preempted/split into pieces which even are allowed to run in parallel time.

For all three problem variants we give a 2-approximate algorithm running in linear time and a 1.5-approximation with near-linear time. For the most complicated case of these three problem contexts, the preemptive case, the presented algorithm improves upon the previously best approximation ratio of $2 - 1/(\lfloor m/2 \rfloor + 1)$ which was given by Monma and Potts [101]. Notice that this ratio is truly greater than 1.5 already if $m \geq 4$.

Chapter 4 SCHEDULING WITH MANY SHARED RESOURCES The setting of scheduling with many shared resources is similar to scheduling problems with batch setup times. The jobs are divided into classes but instead of imposing setup times, the jobs need a shared unit-sized resource, i. e. if a resource is used by a job it cannot be used by another job. Hence, by identifying resources and classes, we simply require that the jobs of each class may not run in parallel time and we aim to find a non-preemptive schedule which minimizes the makespan.

We present a simple $1.\bar{6}$-approximation, a much more involved 1.5-approximation, and approximation schemes. The $1.\bar{6}$-approximation and the 1.5-approximation have better approximation ratios than the previously known $(2m/(m+1))$-approximations by Hebrard et al./Strusevich [61, 113] already for 6 and 4 machines, respectively.

The approximation schemes are two-fold. We provide an EPTAS for a fixed number of machines and an EPTAS with resource augmentation for the general case, i. e. it uses $\lfloor(1+\varepsilon)m\rfloor$ many machines.

1.1.2 *Exact Algorithms*

Chapter 5 RESPONSE TIME ANALYSIS AND MIXING SET In this chapter we connect the fields of real-time scheduling and integer programming. A common setting in real-time scheduling is fixed-priority uniprocessor scheduling. We consider a single processor (or machine) and a system of sporadic tasks which generate jobs periodically. Every task has a given priority and every job of a task has to be processed on the machine before its particular (relative) deadline. The given task priorities directly imply a schedule in the following sense. Whenever a task of higher priority releases a new job, the currently running job (of a task of lower priority) is preempted and the new job is processed instead. Thus, rather than searching for a schedule, one aims to analyze whether a given task system is always schedulable. To this end, a typical approach is the analysis of so-called *worst-case response times* which model these worst-case situations where a job of a lowest-priority task completes at a latest possible point in time. Although sporadic task systems were intensively studied for decades, it was open whether there is a polynomial-time algorithm to compute worst-case response times of harmonic task systems in the presence of so-called task release jitters (which generalize the arrival/release of jobs).

We prove a duality between the computation of worst-case response times, also known as Response Time Computation (RTC), and the MIXING SET problem. Interestingly, both can be modeled as special cases of block-structured ILPs. In more detail, worst-case response times can be computed as a special case of the so-called 4-*block* ILPs, while MIXING SET appears to be a special case of 2-*stage-stochastic* ILPs. To prove this duality we exploit a simple dualization technique which allows to solve inequality-constrained optimization problems in a binary search by consecutively solving a dual formulation of the associated decision problem. For the important case of harmonic periods we prove a Cook reduction from RTC to MIXING SET by using a more

sophisticated algorithm walking over the differences "period minus jitter" in non-decreasing order.

Usually, algorithmic results for worst-case response times are achieved for special cases like equal or harmonic periods only; in fact, we can cope with the general case of *arbitrary* periods. Especially, we give a Turing reduction from RTC to Mixing Set and we study the dependence of the running times on the utilization. We can reverse our methodology to solve the Mixing Set problem by computing worst-case response times for associated real-time task systems. Finally, we show how the dualization technique can by applied to solve specific 4-block ILPs which we prove to be NP-hard to approximate to any constant factor.

FUZZY SIMULTANEOUS CONGRUENCES The complexity of the Mixing Set problem increases with additional upper bound constraints. In this last chapter we generalize over Mixing Set on the one hand and the well-known problem of simultaneous congruences on the other hand. Instead of searching for a positive integer s which is specified by n given remainders modulo integer divisors $a_1, \ldots, a_n$, we consider given *remainder intervals* $R_1, \ldots, R_n$ such that s is feasible if and only if $s \equiv r_i \pmod{a_i}$ holds for *some* remainder r_i in interval R_i for all i. *Chapter 6*

We refer to the problem as Fuzzy Simultaneous Congruences (FSC) and as the Mixing Set problem it is a special case of a 2-stage integer program with only two variables per constraint. We give a hardness result showing that the problem is NP-hard in general by a reduction from Directed Diophantine Approximation. Unlike for Mixing Set, it is already NP-hard to find an *arbitrary* solution to FSC.

The case of *harmonic divisors* was intensively studied for the Mixing Set. Here we investigate harmonic divisors for FSC and we present an algorithm to decide the feasibility of an instance in time $\mathcal{O}(n^2)$ and we show that if it exists even the smallest (or largest) feasible solution can be computed in strongly polynomial time $\mathcal{O}(n^3)$.

Part I

APPROXIMATION ALGORITHMS

2 MAKESPAN MINIMIZATION

2.1 INTRODUCTION

Makespan minimization on identical parallel machines, namely $P||C_{max}$ (cf. Chapter 1), is a widely studied problem both in operations research and in combinatorial optimization and has led to many new algorithmic techniques. The problem is known to be strongly NP-hard and thus we cannot expect to find an exact solution in polynomial time. Many approximation algorithms that run in polynomial time and give a non-optimal solution have been proposed for this problem. In Chapter 1 we already presented some approximation schemes for the problem. However, the running time of such schemes for $P||C_{max}$ were drastically improved over time [7, 68, 91] and the best known running time is $2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))} \log(n) + \mathcal{O}(n)$ due to Jansen and Rohwedder [75] (extension of [74] by the theory of discrepancy), which is subsequently called the JR-algorithm. The JR-algorithm is in fact an algorithm for integer programming, but gives this running time when applied to a natural formulation of $P||C_{max}$. A PTAS with a running time of $f(1/\varepsilon) \cdot n^{\mathcal{O}(1)}$ like in the JR-algorithm is called an efficient polynomial-time approximation scheme (EPTAS).

PTAS's are often believed to be impractical. They tend to yield extremely high (though polynomial) running time bounds even for moderate precisions ε, see Marx [96]. By some, the research on PTAS's has even been considered damaging for the large gap between theory and practice that it creates [109]. Although EPTAS's (when FPTAS's are not available) are sometimes proposed as a potential solution for this situation [96], we are not aware of a practical implementation of an EPTAS.

OUR RESULTS As a major milestone we obtain a generic PTAS implementation that achieves in reasonable time a precision which beats the best known guarantee of a polynomial time non-PTAS algorithm. This precision to the best of our knowledge is $2/11 \approx 18.2\%$, which is guaranteed by the MULTIFIT algorithm. The claim might appear vague, since the running time depends not only on ε, but also on the instance. We believe that it is plausible nevertheless: The algorithm we use, which is based on the JR-algorithm, reduces the problem to performing $\mathcal{O}(\log(n))$ many fast Fourier transformations (FFTs), where the size of the FFT input depends only on ε and not the instance itself. Hence, the running time for all instances (using the same precision) is very stable and predictable. This is in the spirit of an EPTAS running time. We successfully run experiments of our implementation for a precision of $\varepsilon < 2/11$ and thus make the claim that this precision is practically feasible in general. This is also the main message of our result. For completeness, we provide comparisons of the solution quality obtained empirically. While the theoretical guarantee of the PTAS is better, the difference to non-PTAS algorithms is marginal at this state and it is not yet evident in the experiments. The execution of the PTAS

is computationally expensive and the considered precision is on the edge of what is realistic for our implementation. However, we believe that further optimization or more computational resources can lead to also empirically superior results. Nevertheless, the successful execution with a low precision value forms a proof of concept for practical PTAS's.

Towards obtaining such an implementation we need to fine-tune the JR-algorithm significantly. In particular, it requires non-trivial theoretical work and novel algorithmic ideas. In fact, our variant has a slightly better dependence on the precision, namely $2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log\log(1/\varepsilon))}$, giving the best known running time for this problem. Our approach also greatly reduces the constants hidden by the $\mathcal{O}$-notation. We first construct an integer program (IP) — the well-known *configuration IP* — that implies an approximation scheme by rounding the processing times. This IP has properties that allow sophisticated algorithms to solve it efficiently. We present several reduction steps to simplify and compress the IP massively. As extensions of this configuration IP are widely used, we believe this to be of interest in itself. For the makespan minimization problem, we obtain an IP where the columns of the constraint matrix have ℓ_∞-norms bounded by 2 and ℓ_1-norms bounded by $\mathcal{O}(\log(1/\varepsilon))$. In contrast, in the classical configuration integer program used in many of the previous PTAS's both of these norms are bounded by $\mathcal{O}(1/\varepsilon)$. This allows us to greatly reduce the size of the FFT instances in the JR-algorithm without losing the theoretical guarantee. For example, for $\varepsilon \approx 17.29\%$, our reduced IP lowers the instance sizes for FFT from 49^{12} words for the configuration IP to 5^{12} words.

Another important aspect in the algorithm is the rounding of the processing times. In general, one needs to consider only $\mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ different rounded processing times (to guarantee a precision of ε). This number has great impact on the size of the FFT instances. For concrete ε the general rounding scheme might not give the optimal number of rounded processing times. We present a mixed integer linear program that can be used to generically optimize the rounding scheme for guaranteeing a fixed precision ε (or equivalently, for a fixed number of rounded processing times).

RELATED WORK The running time $f(1/\varepsilon) \cdot n^{\mathcal{O}(1)}$ is a fixed-parameter tractable running time, if $1/\varepsilon$ is treated as a parameter. In recent years, the study of practically usable parameterized algorithms has been a growing field of research. This need for practically usable parameterized algorithms has led to the Parameterized Algorithms and Computational Experiments (PACE) challenge [25, 38, 41]. This challenge has brought up surprisingly fast algorithms for important problems such as *treewidth*. Note that the fastest known such algorithm due to Tamaki [114] is based on an algorithm by Bouchitté and Todinca [26], which was widely believed to be purely theoretic. Our work can thus be viewed as an extension of these works to the field of approximation algorithms.

Many approximation algorithms for $P||C_{max}$ were developed over time. The first such algorithm was the longest processing time first (LPT) algorithm by Graham, that achieved approximation ratio 4/3 [55]. In [80], Coffman et al. presented the MULTIFIT algorithm that achieved a better approximation ratio of 13/11. Later it was shown by Yue that this analysis is tight,

i. e. there are instances where MULTIFIT generates a solution with value $13/11 \cdot \mathrm{OPT}$ [117]. Kuruvilla and Palette combined the LPT algorithm and the MULTIFIT algorithm in an iterative way to obtain the Different Job and Machine Sets (DJMS) algorithm [88].

2.2 ALGORITHM

The general idea of our algorithm follows a typical approach for approximation schemes. We follow the dual approximation technique introduced in Chapter 1. Then we simplify the instance such that there are no jobs of very small processing time ($\leq \varepsilon T$) and jobs of very large processing time ($\geq (1-2\varepsilon)T$). The former is standard (see Chapter 1), whereas the latter reduction step is novel. This already reduces the range of processing times significantly for moderate values of ε. The remaining jobs are rounded via a novel rounding to $\mathcal{O}(1/\varepsilon \log(1/\varepsilon))$ different processing times. We can then formulate the problems as an integer program and solve it via the algorithm of Jansen and Rohwedder [75]. Interestingly, our new rounding scheme allows us to compress the well-known configuration integer program quite significantly to obtain a better running time.

2.2.1 *Rounding scheme*

As stated in Chapter 1 it is well known that all jobs j with $p_j \leq \varepsilon T$ can be discarded and added greedily after solving the remaining instance. Let $J_{\text{small}} = \{ j \in J \mid p_j \leq \varepsilon T \}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. Furthermore, we can also get rid of huge jobs J_{huge} with processing times at least $(1-2\varepsilon)T$, as each such job can only be paired with at most one other job from J_{large} without violating the guess T. It is easy to see that we can pair a huge job with the largest possible large job without losing optimality.

Lemma 1. *There is an optimal schedule of J_{large} where each huge job is paired with the largest possible large job (or not paired at all).*

In more detail, define $J_{\text{medium}} = \{ j \in J \mid \varepsilon T < p_j \leq 2\varepsilon T \} \subseteq J_{\text{large}}$ and also assume that $J_{\text{huge}} = \{ j_1, \dots, j_k \}$ with $p_{j_1} \geq \cdots \geq p_{j_k}$ and $J_{\text{medium}} = \{ j'_1, \dots, j'_{k'} \}$ with $p_{j'_1} \geq \cdots \geq p_{j'_{k'}}$. Now, iteratively define a function $\psi \colon [k] \to [k'] \cup \{\infty\}$ with $\psi(j) = \min \{ j' \leq k' \mid p_j + p_{j'} \leq 1 \wedge j' \notin \{ \psi(\tilde{j}) \mid \tilde{j} < j \} \}$ where we set the minimum of an empty set to ∞. Then there is an optimal schedule δ_{large} of J_{large} such that for each machine i with $\delta^{-1}_{\text{large}}(i) \cap J_{\text{huge}} = \{ j \}$, we have either $\delta^{-1}_{\text{large}}(i) = \{ j \}$ if $\psi(j) = \infty$ or $\delta^{-1}_{\text{large}}(i) = \{ j, \psi(j) \}$ if $\psi(j) < \infty$.

Proof. Intuitively, ψ maps j to the largest job j' in J_{medium} with which it can be put onto a machine and which is not already mapped to another huge job. If no such job j' exists, $\psi(j) = \infty$.

Consider any optimal schedule δ_{large} of J_{large} such that there is a machine i with $\delta^{-1}_{\text{large}}(i) = \{ j, j' \}$ and $j \in J_{\text{huge}}$ where either $\psi(j) = \infty$ or $j' \neq \psi(j)$. If there are multiple such machines, we consider the one with the job $j = j_\ell$ with minimal index in the ordering $j_1, \dots, j_k$. As j_ℓ has minimal index, if $\psi(j) = \infty$, all fitting jobs from J_{medium} are packed with the jobs $j_1, \dots, j_{\ell-1}$.

Hence, there is no job j' that can be packed with j_ℓ. This situation is thus not possible. If $j' \neq \psi(j)$, consider the machine $i' = \delta_{\text{large}}(\psi(j))$. As $p_{\psi(j)} + p_j \leq 1$ and $p_{\psi(j)} \geq p_{j'}$ (by definition of $\psi(j)$), we can exchange j' and $\psi(j)$ and still keep a feasible optimal schedule.

Applying the above reason iteratively finally gives us an optimal schedule that always pairs the jobs j and $\psi(j)$ (if $\psi(j) < \infty$). □

As we now know how to place all of the jobs in J_{huge} optimally, we can ignore them and their paired jobs in the following. After removing all of these jobs, we are left with the remaining jobs J_{rem} that we still need to schedule. For all $j \in J_{\text{rem}}$, we now know that we have $p_j \in (\varepsilon T, (1-2\varepsilon)T)$. We will round these remaining item sizes in order to reduce the number of different processing times in our instance. In order to do this, we first split the interval $(\varepsilon T, (1-2\varepsilon)T)$ into $\log(1/\varepsilon)$ growing intervals of size $2^i \varepsilon T$ (starting with $i = 0$). Each of these intervals is then split into $1/\varepsilon$ smaller intervals of the same size.

For example, for $\varepsilon = 1/6$ and $T = 1$, the growing intervals $(1/6, 1/3]$ and $(1/3, 2/3]$ are split into smaller intervals with the following boundaries.

$$\frac{1}{6}, \frac{1}{6}+\frac{1}{36}, \frac{1}{6}+\frac{2}{36}, \frac{1}{6}+\frac{3}{36}, \frac{1}{6}+\frac{4}{36}, \frac{1}{6}+\frac{5}{36},$$
$$\frac{1}{3}, \frac{1}{3}+\frac{1}{18}, \frac{1}{3}+\frac{2}{18}, \frac{1}{3}+\frac{3}{18}, \frac{1}{3}+\frac{4}{18}, \frac{1}{3}+\frac{5}{18}.$$

More formally, for $i \in \mathbb{Z}_{\geq 0}$, let $I_i = (2^i \varepsilon T, 2^{i+1} \varepsilon T]$. In the example above, we thus have $I_0 = (1/6, 1/3]$ and $I_1 = (1/3, 2/3]$. We further partition an intervall I_i into $\lceil 1/\varepsilon \rceil$ subintervals $I_{i,k} = (b_{i,k}, b_{i,k+1}] \cap I_i$ with $b_{i,k} = 2^i \varepsilon T (1 + k/\lceil 1/\varepsilon \rceil)$ for $k \in \{0, \dots, \lceil 1/\varepsilon \rceil - 1\}$. Hence, the above exemplary boundaries are exactly the values $b_{i,k}$ for $i \in \{0, 1\}$ and $k \in \{0, \dots, 5\}$. The processing time of any remaining job $j \in J_{\text{rem}}$ is rounded down to the next lower boundary. We denote this rounded processing time of j by $\tilde{p}_j$.

Lemma 2. *The rounding scheme fulfills the following properties:*

(i) The number of different rounded processing times of J_{rem} is at most $\lceil 1/\varepsilon \rceil \cdot (\log(1/\varepsilon - 2) + 1) \leq \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$.

(ii) A schedule $\tilde{\sigma}$ of the rounded processing times implies a schedule σ of the original processing time with $\mu(\sigma) \leq (1+\varepsilon)\mu(\tilde{\sigma})$.

(iii) We have $b_{i,k_1} + b_{i,k_2} = b_{i+1,(k_1+k_2)/2}$ for all i, k_1, k_2 with $k_1 \equiv k_2 \pmod 2$.

Proof.

(i) We have $2^{i+1}\varepsilon T = (1-2\varepsilon)T$ iff $i+1 = \log(1/\varepsilon - 2)$. For $i > \log(1/\varepsilon - 2)$, there is no job in any interval I_i. Hence, there are at most $\log(1/\varepsilon - 2) + 1$ many values for i such that the processing times lie in I_i. Every I_i is split into $\lceil 1/\varepsilon \rceil$ intervals. Hence, there are at most $\lceil 1/\varepsilon \rceil \cdot (\log(1/\varepsilon - 2) + 1)$ many rounded processing times.

(ii) It is sufficient to show $p_j \geq \tilde{p}_j \geq (1+\varepsilon)^{-1} p_j$. As we only round down, we have $p_j \geq \tilde{p}_j$ immediately. Let $\tilde{p}_j = b_{i,k}$ and hence $p_j \in I_{i,k}$. Then

$$\begin{aligned}
(1+\varepsilon)\tilde{p}_j &= (1+\varepsilon) b_{i,k} \\
&= (1+\varepsilon) 2^i \varepsilon T \Big(1 + \frac{k}{\lceil 1/\varepsilon \rceil}\Big) \\
&= 2^i \varepsilon T \Big(1 + \varepsilon + \frac{(1+\varepsilon)k}{\lceil 1/\varepsilon \rceil}\Big) \\
&\geq 2^i \varepsilon T \Big(1 + \frac{1}{\lceil 1/\varepsilon \rceil} + \frac{k}{\lceil 1/\varepsilon \rceil}\Big) \\
&= 2^i \varepsilon T \Big(1 + \frac{k+1}{\lceil 1/\varepsilon \rceil}\Big) \\
&= b_{i,k+1} > p_j.
\end{aligned}$$

(iii) As $(k_1 + k_2)/2 \in \mathbb{Z}_{\geq 0}$, we have

$$\begin{aligned}
b_{i,k_1} + b_{i,k_2} &= 2^i \varepsilon T \Big(1 + \frac{k_1}{\lceil 1/\varepsilon \rceil}\Big) + 2^i \varepsilon T \Big(1 + \frac{k_2}{\lceil 1/\varepsilon \rceil}\Big) \\
&= 2^i \varepsilon T \Big(2 + \frac{k_1 + k_2}{\lceil 1/\varepsilon \rceil}\Big) \\
&= 2^{i+1} \varepsilon T \Big(1 + \frac{(k_1 + k_2)/2}{\lceil 1/\varepsilon \rceil}\Big) = b_{i+1,(k_1+k_2)/2}.
\end{aligned}$$

This finishes the proof. □

The last property of the Lemma, saying that every two boundaries $b_{i,k}$ and $b_{i,k'}$ of the same interval (with the same parity of k and k') sum up to a boundary $b_{i+1,k''}$ in the next interval, is a crucial ingredient of our rounding scheme and it will be heavily used in the following. Intuitively, this property implies that whenever a job with rounded processing time $b_{i,k}$ and another job with rounded processing time $b_{i,k+2\ell}$ are scheduled on the same machine, we can treat them as a single job with rounded processing time $b_{i+1,k+\ell}$. This allows us to characterize the possible ways to assign rounded jobs to machines in a more compact ways, which in turn allows us to solve the corresponding integer program much faster.

2.2.2 *A new integer program*

The integer program is constructed in the following way. Let d denote the number of rounded item sizes. We will index a vector $x \in \mathbb{Z}^d$ by pairs (i,k), corresponding to the values used in the rounded item sizes $b_{i,k}$ and denote its corresponding entry by $x[i,k]$. A vector $c \in \mathbb{Z}^d_{\geq 0}$ thus describes a possible way to schedule jobs on a machine, where the value $c[i,k]$ describes how many jobs with rounded processing time $b_{i,k}$ are put on a machine. We call such a vector c a *configuration*, if the resulting load of the machine does not exceed T, i. e. $\sum_{i,k} c[i,k] \cdot b_{i,k} \leq T$. Let $\mathcal{C}$ be the set of all configurations. For each $c \in \mathcal{C}$, we have a variable x_c that describes how often configuration c is used, i. e. x_c machines are scheduled according to c. As we only have m

machines available, we are only allowed to use at most m configurations. Hence $\sum_{c\in\mathcal{C}} x_c \leq m$. To guarantee that all jobs are scheduled, let $n_{i,k}$ be the number of items with rounded processing time $b_{i,k}$. Now, summing over all chosen configurations, we want that they contain at least $n_{i,k}$ jobs of rounded processing time $b_{i,k}$. Hence, $\sum_{c\in\mathcal{C}} x_c \cdot c[i,k] \geq n_{i,k}$. Combining these with the natural requirement that $x_c \in \mathbb{Z}_{\geq 0}$, we obtain the following integer program called the *configuration IP*:

$$\begin{aligned} \sum_{c\in\mathcal{C}} x_c &\leq m & & \\ \sum_{c\in\mathcal{C}} x_c \cdot c[i,k] &\geq n_{i,k} & \forall (i,k) & \qquad \text{(confIP)} \\ x_c &\in \mathbb{Z}_{\geq 0} & \forall c \in \mathcal{C} & \end{aligned}$$

The important parameters in the algorithm that we want to use are the number of rows of the constraint matrix (d in our case) and the largest entry in the constraint matrix ($\max_{c\in\mathcal{C},(i,k)}\{c[i,k]\}$). Now, the first property of Lemma 2 already shows that the number of rows of the configuration IP is bounded, i. e. $d \leq \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$. As every boundary $b_{i,k}$ is at least εT and we aim for a maximal load of T, we can easily see that the largest entry of a configuration and thus of the constraint matrix is at most $1/\varepsilon$. A closer look reveals that we actually have the slightly stronger bound of $\|c\|_1 \leq 1/\varepsilon$ for all $c \in \mathcal{C}$. Without jumping too far ahead, the algorithm of Jansen and Rohwedder [75] discussed in Section 2.2.3 will thus yield a running time $2^{\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))} + \mathcal{O}(n)$, which is slightly too high to be usable in practice for our desired approximation guarantee of $\varepsilon < 2/11$. To decrease this running time, we will make use of the last property of Lemma 2, which will give an improved bound of $\|c\|_1 \leq \mathcal{O}(\log(1/\varepsilon))$ and thus improve the running time to $2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log(\log(1/\varepsilon)))} + \mathcal{O}(n)$. Moreover, the hidden constants are significantly lower. This is a sufficient improvement for the algorithm to run in reasonable time for $\varepsilon < 2/11$.

To improve the bound on $\|c\|_1$, we will *add* new columns $\hat{\mathcal{C}}$ to the configuration IP. Remember that Lemma 2 states that for all boundaries $b_{i,k}$ and $b_{i,k'}$ with $k \equiv k' \pmod 2$, there is $b_{i+1,k''} = b_{i,k} + b_{i,k'}$. The main idea behind these new columns $\hat{\mathcal{C}}$ is that whenever we use a job with processing time $b_{i,k}$ and a job with processing time $b_{i,k'}$ on the same machine, we can treat this as a *single job* with processing time $b_{i+1,k''}$. Each new column $\hat{c}(i,k,k')$ will do this exact replacement. The final observation that we need is that in all configurations $c \in \mathcal{C}$ with $\|c\|_1 > 2\log(1/\varepsilon)$, we can do such a replacement: There are only $\log(1/\varepsilon)$ growing large intervals i in our rounding and in each interval we can choose at most two boundaries of different parity $k \not\equiv k' \pmod 2$. Hence, if $\|c\|_1 > 2\log(1/\varepsilon)$, configuration c uses two jobs with processing times $b_{i,k}$ and $b_{i,k'}$ with $k \equiv k' \pmod 2$ and we can thus reduce this configuration via $\hat{c}(i,k,k')$.

By adding the columns $\hat{\mathcal{C}}$ to the configuration integer program, we can remove all configurations c except those in $\mathcal{C}_{\text{red}} = \{c \in \mathcal{C} : \|c\|_1 \leq 2\log(1/\varepsilon)\}$. Let us denote this IP by $\mathsf{IP}_{\mathcal{C}_{\text{red}},\hat{\mathcal{C}}}$. Our discussion above thus implies the following Lemma.

Lemma 3 (informal). *For all solutions to the integer program $\mathsf{IP}_{\mathcal{C}_{\text{red}},\hat{\mathcal{C}}}$, we can compute in linear time a solution to (confIP) and vice versa.*

To prove Lemma 3, we first introduce some notation. For i^*, k_1^*, k_2^* with $k_1^* \equiv k_2^* \pmod 2$ and, we define the new column $\hat{c}(i^*, k_1^*, k_2^*) \in \hat{\mathcal{C}}$ with entries in $\{-1, 0, 1, 2\}$ by

$$\hat{c}(i^*, k_1^*, k_2^*)[i,k] = \begin{cases} 2 & i = i^*,\ k = k_1^* = k_2^* \\ 1 & i = i^*,\ k \in \{k_1^*, k_2^*\},\ k_1^* \neq k_2^* \\ -1 & i = i^* + 1,\ k = (k_1^* + k_2^*)/2 \\ 0 & \text{otherwise} \end{cases}.$$

For a subset $\mathcal{C}' \subseteq \mathcal{C}$, let $\mathsf{IP}_{\mathcal{C}',\hat{\mathcal{C}}}$ be the following integer program:

$$\sum_{c \in \mathcal{C}'} x_c \leq m$$

$$\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} x_c \cdot c[i,k] \geq n_{i,k} \qquad \forall (i,k)$$

$$x_c \in \mathbb{Z}_{\geq 0} \qquad \forall c \in \mathcal{C}' \cup \hat{\mathcal{C}}$$

Let $\mathsf{IP}_{\mathcal{C}}$ be the original configuration IP. We can now obtain the following switching lemma, which directly implies Lemma 3, as a feasible solution x^* of $\mathsf{IP}_{\mathcal{C}}$ can be transformed into a feasible solution y^* of $\mathsf{IP}_{\mathcal{C}_{\text{red}},\hat{\mathcal{C}}}$ and vice versa.

Lemma 4. *Let x^* be a feasible solution of $\mathsf{IP}_{\mathcal{C},\hat{\mathcal{C}}}$.*

1. *If there is a $c_1 \in \mathcal{C}$ with $\|c_1\|_1 > 2(\log(1/\varepsilon - 2) + 1)$ and $x^*_{c_1} > 0$, there are configurations $c_2 \in \mathcal{C}$ with $\|c_2\|_1 < \|c_1\|_1$ and $c_3 \in \hat{\mathcal{C}}$, such that the vector y is a feasible solution of $\mathsf{IP}_{\mathcal{C},\hat{\mathcal{C}}}$ with $y_{c_1} = x^*_{c_1} - 1$, $y_{c_2} = x^*_{c_2} + 1$, $y_{c_3} = 1$ and $y_c = x^*_c$ for all other c.*

2. *If there is a configuration $c_1 = \hat{c}(i^*, k_1^*, k_2^*) \in \hat{\mathcal{C}}$ with $x^*_{c_1} > 0$ and a configuration $c_2 \in \mathcal{C}$ with $c_2[i^* + 1, (k_1^* + k_2^*)/2] = 1$ and $x^*_{c_2} > 0$, there is a configuration $c_3 \in \mathcal{C}$, such that the vector y is a feasible solution of $\mathsf{IP}_{\mathcal{C},\hat{\mathcal{C}}}$ with $y_{c_1} = x^*_{c_1} - 1$, $y_{c_2} = x^*_{c_2} - 1$, $y_{c_3} = x^*_{c_3} + 1$ and $y_c = x^*_c$ for all other c.*

Proof. Let x^* be a feasible solution of $\mathsf{IP}_{\mathcal{C},\hat{\mathcal{C}}}$.

1. As $\|c_1\|_1 > 2(\log(1/\varepsilon - 2) + 1)$, the pidgeonhole principle implies the existence of three indices i^*, k_1^*, k_2^* with $k_1^* \equiv k_2^* \pmod 2$, such that $\min\{c_1[i^*, k_1^*], c_1[i^*, k_2^*]\} \geq 1$, if $k_1^* \neq k_2^*$ or $c_1[i^*, k_1^*] \geq 2$, if $k_1^* = k_2^*$. Choose $c_3 = \hat{c}(i^*, k_1^*, k_2^*)$ and $c_2 = c_1 - c_3$. By construction, we have $c_2 \in \mathcal{C}$ and $\|c_2\|_1 = \|c_1\|_1 + 1 - 2 = \|c_1\|_1 - 1$. For $i = i^*$ and $k = k_1^* = k_2^*$, we have

$$\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} y_c \cdot c[i,k] = \Big(\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} x^*_c \cdot c[i,k] \Big) \underbrace{-2}_{\text{by } c_1} \underbrace{+2}_{\text{by } c_3} \geq n_{i,k}.$$

For $i = i^*$, $k_1^* \neq k_2^*$, and $k \in \{k_1^*, k_2^*\}$, we have

$$\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} y_c \cdot c[i,k] = \Bigg(\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} x_c^* \cdot c[i,k] \Bigg) \underbrace{-1}_{\text{by } c_1} \underbrace{+1}_{\text{by } c_3} \geq n_{i,k}.$$

For $i = i^* + 1$ and $k = (k_1^* + k_2^*)/2$, we have

$$\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} y_c \cdot c[i,k] = \Bigg(\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} x_c^* \cdot c[i,k] \Bigg) \underbrace{+1}_{\text{by } c_2} \underbrace{-1}_{\text{by } c_3} \geq n_{i,k}.$$

Finally, for all other i and k, we have

$$\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} y_c \cdot c[i,k] = \sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} x_c^* \cdot c[i,k] \geq n_{i,k},$$

as nothing changed here.

As $\sum_{c \in \mathcal{C}} y_c = \sum_{c \in \mathcal{C}} x_c^* \leq m$, we can conclude that y is a feasible solution of $\mathsf{IP}_{\mathcal{C},\hat{\mathcal{C}}}$.

2. Choose $c_3 = c_2 - c_1$. By construction, we have $c_3 \in \mathcal{C}$.

   For $i = i^*$ and $k = k_1^* = k_2^*$, we have

$$\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} y_c \cdot c[i,k] = \Bigg(\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} x_c^* \cdot c[i,k] \Bigg) \underbrace{+2}_{\text{by } c_1} \underbrace{-2}_{\text{by } c_3} \geq n_{i,k}.$$

   For $i = i^*$, $k_1^* \neq k_2^*$, and $k \in \{k_1^*, k_2^*\}$, we have

$$\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} y_c \cdot c[i,k] = \Bigg(\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} x_c^* \cdot c[i,k] \Bigg) \underbrace{+1}_{\text{by } c_1} \underbrace{-1}_{\text{by } c_3} \geq n_{i,k}.$$

   For $i = i^* + 1$ and $k = (k_1^* + k_2^*)/2$, we have

$$\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} y_c \cdot c[i,k] = \Bigg(\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} x_c^* \cdot c[i,k] \Bigg) \underbrace{-1}_{\text{by } c_2} \underbrace{+1}_{\text{by } c_3} \geq n_{i,k}.$$

   Finally, for all other i and k, we have

$$\sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} y_c \cdot c[i,k] = \sum_{c \in \mathcal{C}' \cup \hat{\mathcal{C}}} x_c^* \cdot c[i,k] \geq n_{i,k},$$

   as nothing changed here.

   As $\sum_{c \in \mathcal{C}} y_c = \sum_{c \in \mathcal{C}} x_c^* \leq m$, we can conclude that y is a feasible solution of $\mathsf{IP}_{\mathcal{C},\hat{\mathcal{C}}}$.

This finishes the proof. □

The advantage the system $\mathsf{IP}_{\mathcal{C}_{\text{red}},\hat{\mathcal{C}}}$ gives us is that all columns have an ℓ_1-norm bounded by $\mathcal{O}(\log(1/\varepsilon))$. The running time of the JR-algorithm

directly depends on the discrepancy of the underlying constraint matrix. This improved bound on the ℓ_1-norm then allows us to bound this discrepancy leading to a faster running time (see Sec. 2.2.3 for a more thorough discussion). Furthermore, the ℓ_∞-norm of each column is at most 2 (due to the columns in $\hat{\mathcal{C}}$). Already for relatively large values of ε, this reduces the number of columns significantly. For example, for $\varepsilon = 1/6$, the number of columns is reduced from 409 down to 213.

2.2.3 *Applying the JR-algorithm*

Jansen and Rohwedder [75] described an algorithm for integer programming and applied it to the configuration IP for $P||C_{max}$. This algorithm reduces the task of solving the integer program to a small number of fast Fourier transformations (FFTs). The size of the FFT input depends on the number of rows of the constraint matrix as well as its discrepancy. Using the properties of our new integer program we are able to derive much better bounds on the discrepancy of the constraint matrix. Intuitively, the discrepancy of a matrix A measures how well the value $A \cdot (1/2, 1/2, \ldots, 1/2)^\intercal$ can be approximated by the term $Az^\intercal$, where z is some binary vector.

Definition 3 (Discrepancy). *For a matrix $A \in \mathbb{R}^{m \times n}$ the* discrepancy of A *is given as*

$$\operatorname{disc}(A) = \min_{z \in \{0,1\}^n} \left\| A\left(z - \left(\tfrac{1}{2}, \ldots, \tfrac{1}{2}\right)^\intercal\right) \right\|_\infty.$$

Moreover, the hereditary discrepancy *of A is then defined as*

$$\operatorname{herdisc}(A) = \max_{I \subseteq [n]} \operatorname{disc}(A_I)$$

where A_I denotes the matrix A restricted to the columns I.

We sketch the main ideas of the JR-algorithm and refer to [75] for details. The algorithm is based on the idea of splitting the solution to an IP $\{Ax = b, x \in \mathbb{Z}_{\geq 0}\}$ into two parts $x' + x'' = x$ where Ax' and Ax'' are almost the same. Hence, computing all solutions of the IP with $b' \in \mathsf{H}(b/2)$ we can derive a solution with b. Here $\mathsf{H}(b/2)$ is an axis-parallel hypercube with sufficiently large side length surrounding $b/2$. The algorithm then iterates this idea. Indeed, the running time of the algorithm greatly depends on the bound of how evenly a solution can be split, that is, how large the hypercube needs to be. For this, discrepancy is a natural measure. A closer inspection of the JR-algorithm shows that it suffices for their algorithm to take a hypercube of side length $4\operatorname{herdisc}(A) - 1$. Thus, the total number of elements in $\mathsf{H}(b/2^i)$ is at most $(4\operatorname{herdisc}(A))^m$ for all i. The central subprocedure in the algorithm is then to try to combine any two solutions for $b', b'' \in \mathsf{H}(b/2^i)$ to a solution for $b' + b'' \in \mathsf{H}(b/2^{i-1})$. Instead of the naive algorithm that takes quadratic time (in the number of elements of $\mathsf{H}(b/2^i)$) it can be implemented more efficiently as multivariate polynomial multiplication where the input polynomials have m variables and maximum degree $4\operatorname{herdisc}(A) - 1$. This in turn

can be computed efficiently using FFT on inputs of size $\mathcal{O}((4\text{herdisc}(A))^m)$, see Section 2.5 for details.

EVALUATING THE DISCREPANCY The only remaining task now is to bound the discrepancy of our compressed integer program $\mathsf{IP}_{\mathcal{C}_{\text{red}},\hat{\mathcal{C}}}$ presented in Section 2.2.2. Since its columns have small ℓ_1-norm, the classical Beck-Fiala theorem allows us to give a very strong bound on its discrepancy.

Theorem 3 (Beck, Fiala [15]). *For every matrix $A \in \mathbb{R}^{m\times n}$ where the ℓ_1-norm of each column is at most t it holds that* $\text{herdisc}(A) < t$.

Moreover, Bednarchak and Helm [16] observed that for $t \geq 3$ the bound can be improved to $\text{herdisc}(A) \leq t - 3/2$. This can be applied directly to our bounds on the ℓ_1-norm of the configurations of $\mathsf{IP}_{\mathcal{C}_{\text{red}},\hat{\mathcal{C}}}$. Let $A_{\mathcal{C}_{\text{red}},\hat{\mathcal{C}}}$ denote the corresponding constraint matrix. Since for each column c of $A_{\mathcal{C}_{\text{red}},\hat{\mathcal{C}}}$, we have $\|c\|_1 \leq 2\log(1/\varepsilon)$, we get

$$\text{herdisc}(A_{\mathcal{C}_{\text{red}},\hat{\mathcal{C}}}) \leq \mathcal{O}(\log(1/\varepsilon)).$$

Moreover, the number of rows m is one plus the number of rounded processing times, that is, $\mathcal{O}(1/\varepsilon \log(1/\varepsilon))$. The algorithm needs to perform $\mathcal{O}(\log(n))$ many FFTs on input of size $(\log(1/\varepsilon))^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon))}$. Hence, the total running time thus becomes

$$2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log\log(1/\varepsilon))} \log(n) + \mathcal{O}(n).$$

Here $\mathcal{O}(n)$ is necessary for the preprocessing. For a concrete precision ε it makes sense to construct the integer program and determine exactly the maximum ℓ_1-norm of the columns and use this to determine the size of each hypercube $\mathsf{H}(b/2^i)$.

For example, the integer program we derive for precision $\varepsilon \approx 17.29\%$ using the optimized rounding scheme (see next section) gives us a bound of 3 on the ℓ_1-norm and therefore a very moderate bound of 5 on the side length of each hypercube.

The complete algorithm

To sum up the description of our algorithm, we present all steps of the algorithm together in pseudocode in Fig. 2.1. Note that this pseudocode is not optimized (in contrast to our implementation). For example, the computation of the sets J_{small}, J_{large}, and J_{huge} can be done in one sweep and the values $b_{i,k}$ and the rounded processing times $\tilde{p}_j$ can also be computed concurrently. The binary search is performed until lower and upper bound differ by a factor less than $1 + \varepsilon'$. This parameter ε' can be taken negligibly small (in contrast to ε), since the running time grows only logarithmically in ε'.

Optimizing the rounding scheme

Taking a closer look at the proof of Lemma 3 reveals that we only used the last property of Lemma 2 to obtain the simplified integer program. Now,

Input: $I = [p_1, \ldots, p_n, m], \varepsilon, \varepsilon'$

$p_{\max} \leftarrow \max_{j \in [n]} p_j$
$\textsc{lb}(I) \leftarrow \max\{p_{\max}, \sum_{j=1}^{n} p_j / m\}$ ▷ compute bounds
$L \leftarrow \textsc{lb}(I)$
$R \leftarrow 2\,\textsc{lb}(I)$
while $(1+\varepsilon')L < R$ **do** ▷ binary search for OPT
  $T \leftarrow (R+L)/2$ ▷ guess makespan
  $J_{\text{small}} \leftarrow \{j \in [n] \mid p_j \leq \varepsilon T\}$
  $J_{\text{large}} \leftarrow [n] \setminus J_{\text{small}}$
  $J_{\text{huge}} \leftarrow \{j \in J_{\text{large}} \mid p_j \geq (1-2\varepsilon)T\}$
  $J_{\text{large}} \leftarrow J_{\text{large}} \setminus J_{\text{huge}}$
  for $j \in J_{\text{huge}}$ **do**
    find $j' \in J_{\text{large}}$ with $p_{j'}$ minimal and $p_j + p_{j'} \leq T$
    $J_{\text{large}} \leftarrow J_{\text{large}} \setminus \{j'\}$
  for $i = 0, \ldots, \lceil \log(1/\varepsilon - 2) \rceil$ **do**
    for $k = 0, \ldots, \lceil 1/\varepsilon \rceil - 1$ **do**
      $b_{i,k} \leftarrow 2^i \varepsilon T (1 + k/\lceil 1/\varepsilon \rceil)$
      $n_{i,k} \leftarrow 0$
  for $j \in J_{\text{large}}$ **do**
    let $i \in \mathbb{Z}_{\geq 1}$ with $b_{i,k} \leq p_j < b_{i,k+1}$ or $b_{i,\lceil 1/\varepsilon \rceil - 1} \leq p_j < b_{i+1,0}$
    $\tilde{p}_j \leftarrow b_{i,k}$
    $n_{i,k} \leftarrow n_{i,k} + 1$
  construct $\mathsf{IP}_{\mathcal{C}_{\text{red}}, \hat{\mathcal{C}}}$ for $m - |J_{\text{huge}}|$ machines
  solve $\mathsf{IP}_{\mathcal{C}_{\text{red}}, \hat{\mathcal{C}}}$ via the JR-algorithm
  for $j \in J_{\text{huge}}$ **do**
    let j' be the job paired with j if there is one
    assign j and possibly j' to an empty machine
  assign J_{small} greedily
  if $\mathsf{IP}_{\mathcal{C}_{\text{red}}, \hat{\mathcal{C}}}$ has no solution **then**
    $L \leftarrow T$
    continue
  if the makespan exceeds $(1+\varepsilon)T$ **then**
    $L \leftarrow T$
    continue
  $R \leftarrow T$
return the schedule produced for L

Figure 2.1: The non-optimized pseudocode of our algorithm. Here, **continue** means that the current **while**-iteration is aborted and the next iteration is started.

d	optimal precision $\varepsilon(d)$
9	0.172874755859
10	0.160867004395
11	0.15059387207

Table 2.1: Optimal precisions for some numbers of rounded processing times d

we want to find the *best* rounding: Either minimize the number of rounded processing times d for a given ε or minimize the precision ε for a given d. Fortunately, the task to decide whether such a rounding for given d and ε exists can be formulated as a mixed integer program. This allows us to obtain the best rounding in a generic preprocessing step. In the following, x_i will denote the i-th rounded processing time with $x_0 \geq x_1 \geq \cdots \geq x_{d-1}$. Without loss of generality, we assume that our current guess T is equal to 1 here. As our remaining item sizes are in the interval $(\varepsilon, 1-2\varepsilon)$ and we want to obtain a rounding that only produces an error of $1+\varepsilon$, we need to guarantee that

$$\begin{aligned}(1+\varepsilon)x_0 &\geq 1-2\varepsilon,\\ x_0 &\leq 1-2\varepsilon, \text{ and}\\ x_{d-1} &\leq \varepsilon(1+\varepsilon).\end{aligned}$$

Furthermore, we need to make sure that x_i and x_{i+1} are relatively close, i. e $(1+\varepsilon)x_{i+1} \geq x_i$.

To guarantee the last property of Lemma 2, we need to ensure that every configuration with more than $2\log(1/\varepsilon)$ processing times can be reduced. Hence, we construct for every subset $X' \subseteq \{0,\dots,d-1\}$ of size $L+1$ an indicator variable $y_{X'}$ that is 1 iff there is a single configuration containing all of these processing times, i. e. $\sum_{i\in X'} x_i \leq 1$. We also use another indicator variable $z_{i_1,i_2,i}$ that is 1 iff $x_{i_1} + x_{i_2} = x_i$. Now, we need to guarantee that all configurations containing $L+1$ item sizes can be reduced to L item sizes. Hence, if $y_{X'} = 1$ this implies that one can get rid of one of the item sizes, i. e. $z_{i_1,i_2,i} = 1$ for some $i_1, i_2 \in X'$ and $i \in \{0,\dots,d-1\}$. All of these indicator variables and implications can be easily introduced via the big M method or can be directly formulated for the mixed integer program solver. Finally, the x_i are fractional variables and the indicator variables are integral. The complete, formal formulation of the mixed integer program $\mathsf{MIP}_{\varepsilon,d}$ can be found in Section 2.4. Now, we can perform a binary search to optimize either ε or d. More formally, for a given d, we can find the minimal $\varepsilon(d)$, or for a given ε, we can find the minimal $d(\varepsilon)$. See Table 2.1 for the precision achievable with certain values of d.

2.2.4 *Non-PTAS algorithms*

Currently, heuristic algorithms like LPT, the MULTIFIT algorithm [80] and its derivative DJMS [88] give some of the best algorithms to solve instances

of $P||C_{max}$ in practice. For the sake of completeness here we describe briefly how they work.

The MULTIFIT algorithm was presented by Coffman et al. [80] and its exact bound of 13/11 was proved by Yue [117]. It is based on iteratively applying the *First-Fit-Decreasing* algorithm, which, for a given guess on the makespan T, sorts the jobs by their processing times in non-increasing order and the machines in an arbitrary order. Then, each job j is packed onto the first machine where it fits i. e. where the load of T is not exceeded. To approximate an optimum solution of an instance I, the MULTIFIT algorithm takes I and a maximum number of rounds t as the input and starts with computing a lower bound $\ell = \max\{\frac{1}{m}\sum_j p_j, p_1, p_m + p_{m+1}\}$ (where $p_1 \geq \cdots \geq p_n$) and an upper bound $u = \mathrm{LPT}(I)$ to bound the intended makespan. Then MULTIFIT aims to compute the smallest feasible makespan $T^\star \in [\ell, u]$, which admits a First-Fit-Decreasing packing of all jobs to m bins, via binary search in at most t rounds. Obviously, that maximum number of rounds t may be dropped if all numbers in the input are integral. However, it can happen that t rounds are over or even $T^\star = u$ does not admit a feasible First-Fit-Decreasing packing. Then the algorithm returns the solution obtained by LPT.

The DJMS algorithm of Kuruvilla and Paletta [88] combines the LPT approach with the MULTIFIT approach by splitting machines and jobs into *active* and *closed* ones. At the beginning of the algorithm, all jobs and machines are active. Now, in each iteration, the LPT algorithm is applied to the active jobs and active machines to compute an upper bound on the makespan T_{LPT}. Afterwards, MULTIFIT with the upper bound of T_{LPT} is applied to the active jobs and machines. In the solution produced by MULTIFIT, we search for the least loaded active machine $i^\star$ whose load exceeds the lower bound ℓ (as defined for MULTIFIT). All machines with the same load as $i^\star$ and the jobs on them are declared close. These steps are repeated until all machines are closed. In their computational experiments, DJMS gave the best makespan compared with LISTFIT (an algorithm by Gupta and Ruiz-Torres [59]), LPT, and MULTIFIT [88].

2.3 IMPLEMENTATION

The algorithm was implemented in the C++ programming language. To compute fast Fourier transformations we used the C library FFTW3 [50] in version 3.3.8 and we applied OpenMP (https://openmp.org) in version 5.0 for parallelization. Our implementation is publicly available on GitHub [1]. The experiments were computed in the HPC Linux Cluster of Kiel University using 16 cpu cores and 100GB of memory per instance.

2.3.1 *Computational results*

In the following we refer to our algorithm as BDJR. To compare our implementation with the non-PTAS algorithms LPT, MULTIFIT, and DJMS we compute solutions to a set of instances first considered by Kedia [83] in 1971. Since then it has been used by various authors to investigate the quality of scheduling algorithms on identical machines (cf. [58, 88, 89]). The instances are grouped

into four families E1, E2, E3, E4 (see Table 2.2 for an overview). Each family

	m	n	U
E1	$3, 4, 5$	$2m, 3m, 5m$	$[1, 20], [20, 50]$
E2	$2, 3$	$10, 30, 50, 100$	$[100, 800]$
	$4, 6, 8, 10$	$30, 50, 100$	$[100, 800]$
E3	$3, 5, 8, 10$	$3m+1, 3m+2,$ $4m+1, 4m+2,$ $5m+1, 5m+2$	$[1, 100], [100, 200]$
E4	2	10	$[1, 20], [20, 50],$ $[1, 100], [50, 100],$ $[100, 200], [100, 800]$
	3	9	$[1, 20], [20, 50],$ $[1, 100], [50, 100],$ $[100, 200], [100, 800]$

Table 2.2: An overview on the instance families

consists of classes of 100 instances and the instances of each class were generated respecting three common parameters; namely, a number of machines m, a number of jobs n, and a universe interval U, used to uniformly select the integer processing times of the jobs. Overall there are 90 classes (m, n, U) considered, i. e. 9000 instances in total. These instances do not exceed a machine number of $m = 10$, so we generated a family BIG of additional classes where $m \in \{25, 50, 75, 100\}$, $n = 4m$, and $U = [1, 1000]$ (see Table 2.7) to give evidence to the fact that BDJR solves larger instances in reasonable time too. See Figure 2.2 for a makespan comparison of BDJR with the non-PTAS algorithms for class $(m = 4, n = 8, U = [1, 20])$ of family E1 where the 100 instances of the class are presented from left to right while the makespan grows from bottom to top. In Tables 2.3 to 2.7 we give the computation results which are prepared as follows. Each line summarizes the results for the 100 instances of class (m, n, U). Column `better` is the number of instances where the makespan computed by BDJR is lower than the best makespan of the non-PTAS algorithms while `equal` counts the instances where these are equal. Column `avg_q` defined as the quotient of $\sum_{i=1}^{100} \mathrm{BDJR}(i)$ divided by the value $\min\{\sum_{i=1}^{100} \mathrm{LPT}(i), \sum_{i=1}^{100} \mathrm{MF}(i), \sum_{i=1}^{100} \mathrm{DJMS}(i)\}$ compares the sums of the computed makespans (and is rounded to two decimal places) and column `avg_t` states the rounded-up average running time of BDJR in minutes. There were no instance classes, where the maximal running time exceeded $2 \cdot$ `avg_t`.

For each of the classes where `avg_t` $= 1$ the short running time is explained by the rather large quotient n/m causing trivial instances where nearly all jobs are either small ($\leq \varepsilon$) or huge ($\geq 1 - 2\varepsilon$).

By their simple nature the non-PTAS algorithms compute solutions even to large instances within a few seconds. While the running time of the PTAS is generally on a high level, the increase in the running time as the number of machines and jobs grows is rather small due to the parameterized running time of our algorithm. We emphasize that the running time did not cross the border of 5.6 hours for any instance computed. We therefore

family	m	n	U	better	=	avg_q	avg_t
E1	3	6	[1,20]	0	69	1.02	24
E1	3	6	[20,50]	0	22	1.05	44
E1	3	9	[1,20]	2	35	1.03	35
E1	3	9	[20,50]	3	7	1.05	57
E1	3	15	[1,20]	2	42	1.02	42
E1	3	15	[20,50]	2	2	1.04	49
E1	4	8	[1,20]	0	48	1.03	29
E1	4	8	[20,50]	0	11	1.05	49
E1	4	12	[1,20]	1	25	1.04	38
E1	4	12	[20,50]	0	1	1.06	60
E1	4	20	[1,20]	1	23	1.05	46
E1	4	20	[20,50]	0	1	1.06	65
E1	5	10	[1,20]	0	50	1.03	38
E1	5	10	[20,50]	0	5	1.06	60
E1	5	15	[1,20]	0	8	1.06	47
E1	5	15	[20,50]	0	1	1.07	69
E1	5	25	[1,20]	0	7	1.06	51
E1	5	25	[20,50]	0	0	1.07	72

Table 2.3: Computational results for the classes of family E1

family	m	n	U	better	=	avg_q	avg_t
E2	2	10	[100,800]	16	18	1.01	99
E2	2	30	[100,800]	0	58	1	1
E2	2	50	[100,800]	0	56	1	1
E2	2	100	[100,800]	0	66	1	1
E2	3	10	[100,800]	10	7	1.03	125
E2	3	30	[100,800]	2	29	1	1
E2	3	50	[100,800]	0	0	1	1
E2	3	100	[100,800]	0	0	1	1
E2	4	30	[100,800]	7	0	1.03	130
E2	4	50	[100,800]	0	0	1.01	1
E2	4	100	[100,800]	0	38	1	1
E2	6	30	[100,800]	1	0	1.06	145
E2	6	50	[100,800]	6	0	1.02	136
E2	6	100	[100,800]	0	0	1	1
E2	8	30	[100,800]	0	0	1.08	167
E2	8	50	[100,800]	0	0	1.07	162
E2	8	100	[100,800]	0	0	1.01	1
E2	10	30	[100,800]	0	0	1.08	171
E2	10	50	[100,800]	0	0	1.09	164
E2	10	100	[100,800]	0	7	1.01	25

Table 2.4: Computational results for the classes of family E2

see these experiments as a valid proof of concept for practically feasible PTAS's. The empirical solution quality, although not our main focus, is not superior to the non-PTAS algorithms at the considered precision, though the difference is only small. This may be due to the random instances which do not necessarily exhibit a worst-case structure and that the difference in

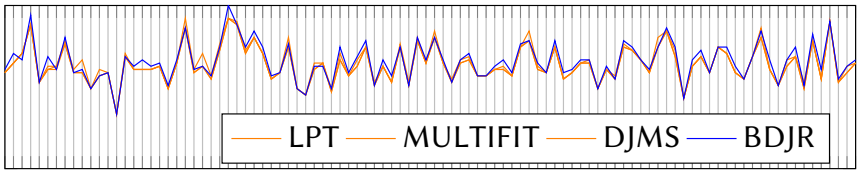


Figure 2.2: Makespan comparison for 100 instances of class $(3, 8, [1, 20])$ (x: instances, y: makespan)

family	m	n	U	better	=	avg_q	avg_t
E3	10	31	[100,200]	0	0	1.07	182
E3	10	31	[1,100]	0	0	1.08	82
E3	10	32	[100,200]	0	2	1.07	179
E3	10	32	[1,100]	0	0	1.09	81
E3	10	41	[100,200]	0	0	1.07	214
E3	10	41	[1,100]	0	0	1.09	80
E3	10	42	[100,200]	0	0	1.07	220
E3	10	42	[1,100]	0	0	1.1	82
E3	10	51	[100,200]	0	0	1.08	225
E3	10	51	[1,100]	0	0	1.1	80
E3	10	52	[100,200]	0	0	1.06	224
E3	10	52	[1,100]	0	0	1.09	75
E3	3	10	[100,200]	4	15	1.03	114
E3	3	10	[1,100]	3	21	1.03	67
E3	3	11	[100,200]	21	2	1.02	150
E3	3	11	[1,100]	4	20	1.03	61
E3	3	13	[100,200]	20	7	1.03	133
E3	3	13	[1,100]	4	15	1.03	60
E3	3	14	[100,200]	20	1	1.03	146
E3	3	14	[1,100]	6	7	1.03	62
E3	3	16	[100,200]	30	2	1.02	166
E3	3	16	[1,100]	9	13	1.03	67
E3	3	17	[100,200]	7	1	1.03	146
E3	3	17	[1,100]	5	11	1.03	59
E3	5	16	[100,200]	1	1	1.05	135
E3	5	16	[1,100]	0	3	1.06	66
E3	5	17	[100,200]	8	2	1.03	149
E3	5	17	[1,100]	2	4	1.05	68
E3	5	21	[100,200]	19	3	1.02	184
E3	5	21	[1,100]	1	0	1.06	65
E3	5	22	[100,200]	4	2	1.04	178
E3	5	22	[1,100]	1	1	1.06	63
E3	5	26	[100,200]	11	1	1.04	185
E3	5	26	[1,100]	2	2	1.06	67
E3	5	27	[100,200]	1	0	1.05	166
E3	5	27	[1,100]	2	1	1.05	67
E3	8	25	[100,200]	1	0	1.06	176
E3	8	25	[1,100]	0	0	1.07	74
E3	8	26	[100,200]	0	0	1.06	177
E3	8	26	[1,100]	0	0	1.07	81
E3	8	33	[100,200]	0	1	1.06	202
E3	8	33	[1,100]	0	0	1.09	72
E3	8	34	[100,200]	1	0	1.06	195
E3	8	34	[1,100]	0	0	1.08	78
E3	8	41	[100,200]	0	0	1.06	207
E3	8	41	[1,100]	0	0	1.08	76
E3	8	42	[100,200]	0	0	1.06	211
E3	8	42	[1,100]	0	0	1.08	77

Table 2.5: Computational results for the classes of family E3

family	m	n	U	better	=	avg_q	avg_t
E_4	2	10	[1,20]	4	61	1.01	39
E_4	2	10	[1,100]	14	21	1.01	48
E_4	2	10	[20,50]	10	12	1.02	46
E_4	2	10	[50,100]	14	9	1.03	59
E_4	2	10	[100,200]	15	3	1.02	71
E_4	2	10	[100,800]	14	8	1.01	97
E_4	3	9	[1,20]	0	43	1.03	37
E_4	3	9	[1,100]	6	29	1.03	57
E_4	3	9	[20,50]	2	9	1.05	61
E_4	3	9	[50,100]	2	0	1.05	75
E_4	3	9	[100,200]	2	0	1.05	95
E_4	3	9	[100,800]	5	14	1.03	132

Table 2.6: Computational results for the classes of family E_4

family	m	n	U	better	=	avg_q	avg_t
BIG	25	100	[1,1000]	0	0	1.13	171
BIG	50	200	[1,1000]	0	0	1.14	172
BIG	75	300	[1,1000]	0	0	1.14	179
BIG	100	400	[1,1000]	0	0	1.15	180

Table 2.7: Computational results for the classes of family BIG

theoretical approximation guarantee is at this state fairly small. We believe that with additional computational resources or further optimizations an even lower precision can be reached, which will lead to a superior solution quality also in experiments.

2.4 THE COMPLETE MILP TO OPTIMIZE THE ROUNDING SCHEME

The y-variables can be easily introduced via the big M method and by minimizing the y variables:

$$-\sum_{i \in X'} x_i + M \cdot y_{X'} \leq -1 \quad \forall X'$$
$$(y_{X'} = 1 \Leftrightarrow \sum_{i \in X'} x_i \leq 1),$$

where $X' \subseteq \{0, \ldots, d-1\} \wedge |X'| = L+1$. We also construct another indicator variable $z_{i_1,i_2,i}$ that is 1 iff $x_{i_1} + x_{i_2} = x_i$ holds.

$$z_{i_1,i_2,i} = 1 \Leftrightarrow x_{i_1} + x_{i_2} = x_i \qquad \forall i_1, i_2, i \in \{0, \ldots, d-1\}$$

Again, this implication can be formulated via the big M method or directly by a mixed integer program solver.

Now, we need to guarantee that all configurations containing $L+1$ item sizes can be reduced to L item sizes. Hence, if $y_{X'} = 1$ this implies that one

can get rid of one of the item sizes, i. e. $z_{i_1,i_2,i} = 1$ for some $i_1, i_2 \in X'$ and $i \in \{0,\dots,d-1\}$:

$$y_{X'} = 1 \implies \\ \exists i_1', i_2' \in X' \exists i \in \{0,\dots,r-1\} : z_{i_1',i_2',i} = 1 \quad \forall X',$$

where $X' \subseteq \{0,\dots,d-1\} \wedge |X'| = L+1$ Again, this implication can be formulated via the big M method or directly by an mixed integer program solver. Finally, the x_i are fractional variables and the indicator variables are integral:

$$\begin{aligned} x_i &\in [0,1] && \forall i \in \{0,\dots,d-1\} \\ y_{X'} &\in \{0,1\} && \forall X' \subseteq \{0,\dots,d-1\} \wedge |X'| = L+1 \\ z_{i_1,i_2,i} &\in \{0,1\} && \forall i_1, i_2, i \in \{0,\dots,d-1\} \end{aligned}$$

Together, we obtain the following mixed integer program:

$$\begin{aligned} \min \sum_{X'} y_{X'} + \sum_{i_1,i_2,i} z_{i_1,i_2,i} \\ (1+\varepsilon)x_0 &\geq 1-2\varepsilon \\ x_0 &\leq 1-2\varepsilon \\ x_{d-1} &\leq \varepsilon \cdot (1+\varepsilon) \\ (1+\varepsilon)x_{i+1} &\geq x_i \quad \forall i = 1,\dots,d-2 \\ y_{X'} = 1 &\Leftrightarrow \sum_{i \in X'} x_i \leq 1 \\ z_{i_1,i_2,i} = 1 &\Leftrightarrow x_{i_1} + x_{i_2} = x_i \quad \forall i_1, i_2, i \in \{0,\dots,d-1\} \\ y_{X'} = 1 &\Rightarrow \\ &\exists i_1', i_2' \in X' \exists i \in [n] : z_{i_1',i_2',i} = 1 \quad \forall X' \\ x_i &\in [0,1] \quad \forall i \in \{0,\dots,d-1\} \\ y_{X'} &\in \{0,1\} \quad \forall X' \\ z_{i_1,i_2,i} &\in \{0,1\} \quad \forall i_1, i_2, i \in \{0,\dots,d-1\}, \end{aligned}$$

where $X' \subseteq \{0,\dots,d-1\} \wedge |X'| = L+1$.

2.5 COMPUTING MULTIDIMENSIONAL CONVOLUTIONS WITH FFT

The JR-algorithm depends on multiplying multi-variate polynomials. Here we describe how to do this using the well-known fast Fourier transformation (FFT). We start by introducing multidimensional polynomials as well as multidimensional discrete Fourier transformations (DFTs).

Definition 4. *A* multivariate *or* multidimensional *polynomial p of d variables $x_1,\dots,x_d$ and coordinate degree $n = (n_1,\dots,n_d)$ is a linear combination of monomials, i. e. $p(x) = \sum_{k \leq n-\mathbf{1}} p_k x^k$ where $x^k = \prod_{j=1}^{d} x_j^{k_j}$ for all $k \in \bigtimes_{j=1}^{d} \{0,\dots,n_j-1\}$, $n-\mathbf{1} = (n_1-1,\dots,n_d-1)$, and $p_k \in \mathbb{C}$ for all $k \leq n-\mathbf{1}$. So, p is a (univariate) polynomial of degree n_j in each coordinate direction j.*

Definition 5. *For two multivariate polynomials* $f(x) = \sum_{k \le n-\mathbf{1}} f_k x^k$ *and* $g(x) = \sum_{k \le n-\mathbf{1}} g_k x^k$ *the multidimensional discrete convolution* $f * g$ *is defined by* $(f * g)(x) = \sum_{k \le 2n-2} c_k x^k$ *where* $c_k = \sum_{j \le k} f_j \cdot g_{k-j}$*, i. e.*

$$c_{(k_1,\dots,k_d)} = \sum_{j_1=0}^{k_1} \sum_{j_2=0}^{k_2} \cdots \sum_{j_d=0}^{k_d} f_{(j_1,\dots,j_d)} \cdot g_{(k_1-j_1,\dots,k_d-j_d)}.$$

Hence, a truly primitive approach to compute the convolution $f * g$ takes time $\mathcal{O}(\sum_{k \le 2n-2} \prod_{j=1}^{d} (k_j + 1)) \le \mathcal{O}(2^d N^2)$ where $N = \prod_{j=1}^{d} n_j$ is the number of all input points.

However, the convolution theorem says that $\mathcal{F}\{f * g\} = N \cdot \mathcal{F}\{f\} \odot \mathcal{F}\{g\}$ where $\mathcal{F}$ denotes the Fourier transform operator and $\odot$ denotes the pointwise multiplication, i. e. $(v \odot w)_k = v_k \cdot w_k$. Therefore, one can compute the convolution $f * g$ by computing

$$f * g = \mathcal{F}^{-1}\{N \cdot \mathcal{F}\{f\} \odot \mathcal{F}\{g\}\}.$$

For our goals we only depend on the *discrete* Fourier transformation as follows.

Definition 6 (DFT)**.** *The multidimensional discrete Fourier transformation (DFT)* $\hat{f} = \mathcal{F}\{f\}$ *of* f *is defined by*

$$\hat{f}_k = \sum_{\ell \le n-\mathbf{1}} f_\ell \cdot \exp\Big(-i2\pi \sum_{j=1}^{d} \frac{k_j \ell_j}{n_j}\Big) \quad \forall k \le n - \mathbf{1}$$

whereas the inverse *multidimensional DFT* $\mathcal{F}^{-1}\{\hat{f}\}$ *of* $\hat{f}$ *is given by*

$$f_k = \frac{1}{N} \sum_{\ell \le n-\mathbf{1}} \hat{f}_\ell \cdot \exp\Big(i2\pi \sum_{j=1}^{d} \frac{k_j \ell_j}{n_j}\Big) \quad \forall k \le n - \mathbf{1}.$$

To give more light to these definitions let us consider the 1-dimensional case, i. e. $d = 1, n = n_1$. Then Definition 6 simplifies to

$$\hat{f}_k = \sum_{\ell=0}^{n-1} f_\ell \cdot e^{-i2\pi k\ell/n}$$

$$f_k = \frac{1}{n} \sum_{\ell=0}^{n-1} \hat{f}_\ell \cdot e^{i2\pi k\ell/n} \quad \forall k = 0, \dots, n-1.$$

Obviously, since $\hat{f}_k = f(e^{-i2\pi k/n})$ for $k = 0, \dots, n-1$ one can compute the DFT of f in time $\mathcal{O}(n^2)$ by simply evaluating f in n points. Fortunately, this running time can be reduced to $\mathcal{O}(n \log n)$ by algorithmically exploiting the fact, that the evaluation of

$$\begin{aligned} f(x) &= (f_0 x^0 + f_2 x^2 + \dots) + x \cdot (f_1 x^0 + f_3 x^2 + \dots) \\ &= p(x^2) + x \cdot q(x^2) \end{aligned}$$

can always be written by the evaluation of two polynomials p, q of half-sized degrees for any input point x. This running time improvement is the reason to call a DFT an FFT. Also the inverse can be computed efficiently by using just the same trick, since $f_k = \frac{1}{n}\hat{f}(e^{i2\pi k/n})$. See Fig. 2.3 for an illustration of the whole computation.

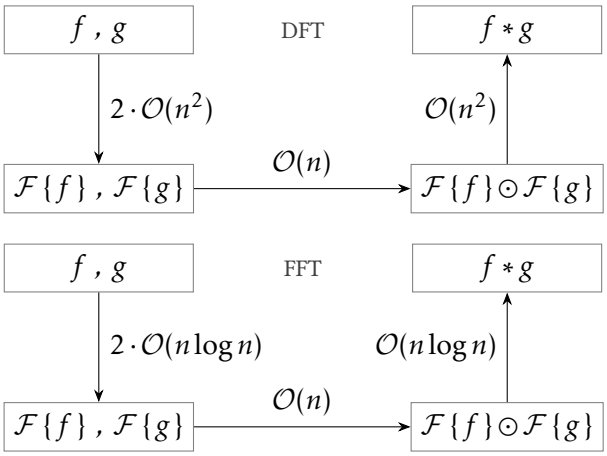


Figure 2.3: Computing a 1D-convolution with a discrete/fast Fourier transformation

Finally, turning back to the general case $n = (n_1, \ldots, n_d)$ we can compute the multidimensional FFT of f by computing 1-dimensional FFTs. In more detail, we may divide the whole computation into N/n_j many 1-dimensional transformations of size n_j for each coordinate direction j. This yields the usual running time of

$$\sum_{j=1}^{d} \frac{N}{n_j} \mathcal{O}(n_j \log n_j) = \mathcal{O}\left(N \sum_{j=1}^{d} \log n_j\right) = \mathcal{O}(N \log N).$$

2.6 OPEN QUESTIONS

The presented results do not lie at the end of the road. On the practical side, this work is a proof of concept and our implementation holds a great potential to be improved to achieve even better running times in practice.

On the theoretical side, there still is a gap between the sharpest known bounds for the running time of approximation schemes for $P\|C_{max}$. Our algorithm gives the best known upper bound of $2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log\log(1/\varepsilon))} + \mathcal{O}(n)$ while the strongest known lower bound of Chen, Jansen, and Zhang [28] says that for any $\delta < 1$, there is no EPTAS for $P\|C_{max}$ running in time $2^{\mathcal{O}((1/\varepsilon)^{\delta})} + n^{\mathcal{O}(1)}$, unless the Exponential Time Hypothesis (ETH) fails. To close this gap remains as an intriguing open question.

3

SCHEDULING WITH BATCH SETUP TIMES

3.1 INTRODUCTION

Scheduling problems with setup times have been intensively studied for decades; in fact, they allow very natural formulations of scheduling problems.

In the general scheduling problem with setup times, there are m identical and parallel machines, a set J of $n \in \mathbb{Z}_{\geq 1}$ jobs $j \in J$, $c \in \mathbb{Z}_{\geq 1}$ different classes, a partition $\dot{\bigcup}_{i=1}^{c} C_i = J$ of c nonempty and disjoint subsets $C_i \subseteq J$, a *processing time* of $t_j \in \mathbb{Z}_{\geq 1}$ time units for each job $j \in J$ and a *setup* (or *setup time*) of $s_i \in \mathbb{Z}_{\geq 1}$ time units for each class $i \in [c]$. The objective is to find a schedule which minimizes the makespan while holding all of the following.

All jobs (or its complete sets of job pieces) are scheduled. A setup s_i is scheduled whenever a machine starts processing load of class i and when switching processing from one class to another *different* class on a machine. A setup is *not* required between jobs (or job pieces) of the *same* class. There are various types of setups discussed; here we focus on *sequence-independent* batch setups, i. e. they do not depend on the previous job/class. All machines are *single-threaded* (jobs (or job pieces) and setups do not intersect in time on each machine) and no setup is preempted.

There are three variants of scheduling problems with setup times which have been gaining the most attention in the past. There is the *non-preemptive* case where no job may be preempted, which is also formally known as the problem $\mathrm{P}|\mathrm{setup}=\mathrm{s_i}|\mathrm{C_{max}}$. Another variant is the *preemptive* context, namely $\mathrm{P}|\mathrm{pmtn},\mathrm{setup}=\mathrm{s_i}|\mathrm{C_{max}}$, where a job may be preempted at any time but be processed on at most one machine at a time, so a job may not be parallelized. In the generous case of *splittable* scheduling, known as $\mathrm{P}|\mathrm{split},\mathrm{setup}=\mathrm{s_i}|\mathrm{C_{max}}$, a job is allowed to be split into any number of job pieces which may be processed on any machine at any time.

RELATED RESULTS Monma and Potts began their investigation of these problems considering the preemptive case. They found first dynamic programming approaches for various single machine problems [100] polynomial in n but exponential in c. Furthermore, they showed NP-hardness for

	m variable		m fixed
	unrestricted	*small batches* or $\lvert C_i\rvert = 1$ or $P(C_i) \leq \gamma\,\mathrm{OPT}$	
Splittable	5/3 in poly [116] 3/2 in $\mathcal{O}(n + c\log(c+m))$ * EPTAS [67]	$\approx \frac{3}{2}$ in $\mathcal{O}(n + (m+c)\log(m+c))$ [27]	FPTAS [116]
Non-Preemptive	$2+\varepsilon$ in $\mathcal{O}(n \log 1/\varepsilon)$, PTAS [69] 3/2 in $\mathcal{O}(n\log(n+\Delta))$ * EPTAS [67]	$(1+\varepsilon)\min\{\frac{3}{2}\mathrm{OPT}, \mathrm{OPT}+t_{\max}-1\}$ in poly [95]	FPTAS [95]
Preemptive	$(2-(\lfloor m/2\rfloor+1)^{-1})$ in $\mathcal{O}(n)$ [101] 3/2 in $\mathcal{O}(n\log n)$ *	4/3 + ε in poly [110] EPTAS [67]	open

Table 3.1: An overview of known approximation results * Result is in this chapter

$P|pmtn, setup = s_i|C_{max}$ even if $m = 2$. In a later work [101] they found a heuristic which resembles McNaughton's preemptive wrap-around rule [97]. It requires $\mathcal{O}(n)$ time for being $(2 - (\lfloor \frac{m}{2} \rfloor + 1)^{-1})$-approximate. Notice that this ratio is truly greater than 3/2 if $m \geq 4$ and the asymptotic bound is 2 for $m \to \infty$. Monma and Potts also discussed the problem class of *small batches* where for any batch i the sum of one setup time and the total processing time of all jobs in i is smaller than the optimal makespan, i. e. $s_i + \sum_{j \in C_i} t_j \leq \mathrm{OPT}$. Most suitable for this kind of problems, they found a heuristic that first uses list scheduling for complete batches followed by an attempt of splitting some batches so that they are scheduled on two different machines. This second approach needs a running time of $\mathcal{O}(n + (m+c)\log(m+c))$ and considering only small batches it is $(1.5 - \frac{1}{4m-4})$-approximate if $m \leq 4$ whereas it is $(1.\bar{6} - \frac{1}{m})$-approximate for small batches if m is a multiple of 3 and $m \geq 6$.

Then Chen [27] modified the second approach of Monma and Potts. For small batches Chen improved the heuristic to a worst case guarantee of $\max\{\frac{3m}{2m+1}, \frac{3m-4}{2m-2}\}$ if $m \geq 5$ while the same time of $\mathcal{O}(n + (m+c)\log(m+c))$ is required.

Schuurman and Woeginger [110] studied the preemptive problem for *single-job-batches*, i. e. $|C_i| = 1$. They found a PTAS for the uniform setups problem $P|pmtn, setup = s|C_{max}$. Furthermore, they presented a $(1.\bar{3} + \varepsilon)$-approximation in case of arbitrary setup times. Both algorithms have a running time linear in n but exponential in $1/\varepsilon$. Then Chen, Ye, and Zhang [116] turned to the splittable case. Without other restrictions they presented an FPTAS if m is fixed and a $1.\bar{6}$-approximation in polynomial time if m is variable. They give simple arguments that the problem is weakly NP-hard if m is fixed and strongly NP-hard otherwise.

More recently Mäcker et al. [95] made progress to the case of non-preemptive scheduling. They used the restrictions that all setup times are equal and the total processing time of each class is bounded by $\gamma\,\mathrm{OPT}$ for some constant γ, i. e. $\sum_{j \in C_i} t_j \leq \gamma\,\mathrm{OPT}$. They found a simple 2-approximation, an FPTAS for fixed m, and a $(1+\varepsilon)\min\{\frac{3}{2}\mathrm{OPT}, \mathrm{OPT} + t_{max} - 1\}$-approximation (where $t_{max} = \max_{j \in J} t_j$) in polynomial time if m is variable. Therefore, this especially yields a PTAS for unit processing times $t_j = 1$.

Jansen and Land [69] found three different algorithms for the non-preemptive context without restrictions. They presented an approximation ratio 3 using a next-fit strategy running in time $\mathcal{O}(n)$, a 2-dual approximation running in time $\mathcal{O}(n)$ which leads to a $(2+\varepsilon)$-approximation running in time $\mathcal{O}(n\log(\frac{1}{\varepsilon}))$, as well as a PTAS. Jansen et al. [67] found an EPTAS using n-fold ILPs for all three problem variants. For the preemptive case they assume $|C_i| = 1$. However, the running times are highly exponential in $1/\varepsilon$. These algorithms give important answers to the question of complexity but they are rather useless for solving actual problems in practice. Therefore the design of *fast* (and especially polynomial-time) approximation algorithms with small approximation ratio remains interesting.

OUR CONTRIBUTION For all three problem variants we give a 2-approximate algorithm running in time $\mathcal{O}(n)$ as well as a $(1.5+\varepsilon)$-approximation with running time $\mathcal{O}(n\log(1/\varepsilon))$. With some runtime improvements we present

some very efficient near-linear approximation algorithms with a constant approximation ratio equal to 1.5. In detail, we find a 1.5-approximation for the splittable case with running time $\mathcal{O}(n + c\log(c+m)) \leq \mathcal{O}(n\log(c+m))$. Also we will see a 1.5-approximate algorithm for the non-preemptive case that runs in time $\mathcal{O}(n\log(T_{\min}))$ where $T_{\min} = \max\{\frac{1}{m}N, \max_{i\in[c]}(s_i + t^{(i)}_{\max})\}$, $t^{(i)}_{\max} = \max_{j\in C_i} t_j$ and $N = \sum_{i=1}^{c} s_i + \sum_{j\in J} t_j$. For the most complicated case of these three problem contexts, the preemptive case, we study a 1.5-approximation running in time $\mathcal{O}(n\log(c+m)) \leq \mathcal{O}(n\log n)$. Especially this last result is interesting; we make progress to the general case where classes may consist of an *arbitrary* number of jobs. The best approximation ratio was the one by Monma and Potts [101] mentioned above. All other previously known results for preemptive scheduling used restrictions like *small batches* or even *single-job-batches*, i. e. $|C_i| = 1$ (cf. Table 3.1). As a byproduct we give some new *dual* lower bounds.

ALGORITHMIC IDEAS The 1.5-approximate algorithm for the preemptive case is our main result. It is highly related to the right partitioning of classes and jobs into different sizes; in fact, the right partition allows us to reduce the problem to a fine-grained knapsack instance. To achieve the truly constant bounds in the splittable and preemptive case *while* speeding up the algorithm we use a technique that we call *Class Jumping* (see Sections 3.3.4 and 3.6.4). However, we also make extensive use of a simple idea that we name *Batch Wrapping* (see Section 3.2.1).

3.2 PRELIMINARIES

NOTATION The *load* of a machine $u \in [m]$ in a schedule σ is $\mathrm{L}_\sigma(u)$ (or simply $\mathrm{L}(u)$). This is the sum of all setup times and the processing times of all jobs (or job pieces) scheduled on machine u. The processing time of a set of jobs K is $P(K) = \sum_{j\in K} t_j$. The jobs of a set of classes $X \subseteq [c]$ are $J(X) = \bigcup_{i\in X} C_i$. A *job piece* of a job $j \in J$ is a (new) job j' with a processing time $t_{j'} \leq t_j$. Whenever we split a job $j \in C_i$ of a class $i \in [c]$ into two new job pieces j_1, j_2, we understand these jobs to be jobs of class i as well - although $j_1, j_2 \in C_i$ does not hold formally.

PROPERTIES For later purposes we need to split the classes into expensive classes and cheap classes as follows. Let $T > 0$ be a makespan. We say that a class $i \in [c]$ is *expensive* if $s_i > \frac{1}{2}T$ and we call it *cheap* if $s_i \leq \frac{1}{2}T$. We define $I_{\exp} \subseteq [c]$ as the set of all expensive classes and $I_{\mathrm{chp}} \subseteq [c]$ as the set of all cheap classes such that $I_{\exp} \dot\cup I_{\mathrm{chp}} = [c]$. We denote the total load of a feasible schedule σ by $\mathrm{L}(\sigma) = \sum_{u=1}^{m} \mathrm{L}_\sigma(u) = \sum_{i=1}^{c} (\lambda_i^\sigma s_i + P(C_i))$ for some setup multiplicities $\lambda_i^\sigma \in \mathbb{Z}_{\geq 1}$ with $\lambda_i^\sigma \leq |C_i|$. For any instance I it is true that $\mathrm{OPT}(I) \leq N := \sum_{i\in[c]} s_i + \sum_{j\in J} t_j$ (all jobs on one machine) as well as $\mathrm{OPT}(I) > s_{\max}$ and $\mathrm{OPT}(I) \geq \frac{1}{m}N$ and therefore $\mathrm{OPT}(I) \geq \max\{\frac{1}{m}N, s_{\max}\}$.

An important value to our observations will be the minimal number of machines to schedule all jobs of an expensive class. In the following we give two simple lemmas to find this minimal machines numbers. Therefore, for all classes $i \in [c]$ let $\alpha_i = \lceil P(C_i)/(T - s_i)\rceil$ and $\beta_i = \lceil 2P(C_i)/T\rceil$.

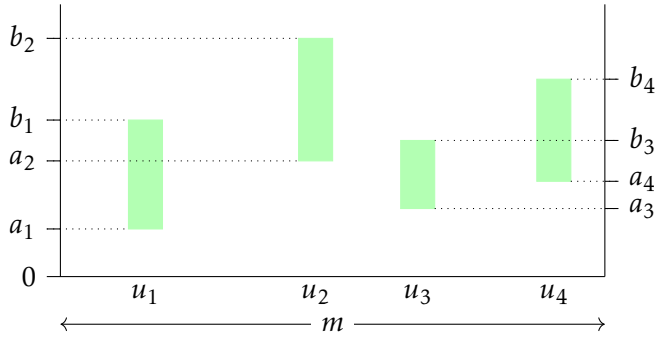


Figure 3.1: An example of a wrap template ω with $|\omega| = 4$

Lemma 5. *Given a feasible schedule σ with makespan T and load $\mathrm{L}(\sigma) = \sum_{i=1}^{c}(\lambda_i^\sigma s_i + P(C_i))$, it is true that $\lambda_i^\sigma \geq \alpha_i \geq 1$. Furthermore, $i \in I_{\exp}$ implies that $\lambda_i^\sigma \geq \alpha_i \geq \beta_i \geq 1$ and σ needs at least λ_i^σ different machines to place all jobs in C_i.*

Proof. Apparently $\alpha_i \geq 1$ and $\beta_i \geq 1$ are direct results for all $i \in [c]$. There must be at least one initial setup time s_i to schedule any jobs of class i on a machine. Since setups may not be split, there is a processing time of at most $T - s_i$ per machine to schedule jobs of C_i and therefore, σ needs at least $\alpha_i = \lceil P(C_i)/(T - s_i) \rceil \leq \lambda_i^\sigma$ setups to schedule all jobs of C_i. If $i \in I_{\exp}$ we have $s_i > \frac{1}{2}T$ such that there cannot be two expensive setups on one machine and $\alpha_i = \lceil P(C_i)/(T - s_i) \rceil \geq \lceil P(C_i)/(T - \frac{1}{2}T) \rceil = \lceil 2P(C_i)/T \rceil = \beta_i$. □

Lemma 6. *Let σ be a feasible schedule with makespan T for an instance I. Then σ schedules jobs of different expensive classes on different machines and $m \geq \sum_{i \in I_{\exp}} \lambda_i^\sigma$.*

Proof. Assume that $m < \sum_{i \in I_{\exp}} \lambda_i^\sigma$. Then setups of two different classes $i_1, i_2 \in I_{\exp}$ must have been scheduled on one machine $u \in [m]$. We obtain $\mathrm{L}_\sigma(u) \geq s_{i_1} + s_{i_2} > \frac{1}{2}T + \frac{1}{2}T = T$ since i_1 and i_2 are expensive. That is a contradiction to the makespan T. □

3.2.1 *Batch Wrapping*

Robert McNaughton gave an algorithm to solve $\mathrm{P}|\mathrm{pmtn}|\mathrm{C}_{\max}$ in linear time [97]. McNaughton's *wrap-around rule* simply schedules all jobs greedily from time 0 to time $T = \max\{t_{\max}, \frac{1}{m}\sum_{j \in J} t_j\}$ splitting jobs whenever they cross the border T. Indeed, this is not applicable for our setup time problems. However, our idea of *Batch Wrapping* can be understood as a generalization of McNaughton's wrap-around rule suitable for scheduling with setup times. To define it we need *wrap templates* and *wrap sequences* as follows.

Definition 7. *A* wrap template *is a list $\omega = (\omega_1, \ldots, \omega_{|\omega|}) \in ([m] \times \mathbb{Q} \times \mathbb{Q})^*$ of triples $\omega_r = (u_r, a_r, b_r) \in [m] \times \mathbb{Q} \times \mathbb{Q}$ for $1 \leq r \leq |\omega|$ that hold the following properties:*

$$(i)\ u_r < u_{r+1}\ \text{f.a.}\ 1 \leq r < |\omega| \quad (ii)\ 0 \leq a_r < b_r\ \text{f.a.}\ 1 \leq r \leq |\omega|$$

Let $S(\omega) = \sum_{r=1}^{|\omega|}(b_r - a_r)$ *denote the provided period of time. A* wrap sequence *is a sequence* $Q = [s_{i_l}, C'_l]_{l\in[k]}$ *where*

$$[s_{i_l}, C'_l]_{l\in[k]} = (s_{i_1}, j_1^1, \ldots, j_{n_1}^1, s_{i_2}, j_1^2, \ldots, j_{n_2}^2, \ldots, s_{i_k}, j_1^k, \ldots, j_{n_k}^k)$$

and $C'_l = \{j_1^l, \ldots, j_{n_l}^l\}$ *with* $n_l = |C'_l|$ *is a set of jobs and/or job pieces of a class* $i_l \in [c]$ *for* $1 \le l \le k$. *Let* $\mathrm{L}(Q) = \sum_{l=1}^{k}(s_{i_l} + P(C'_l))$ *denote the load of* Q.

These technical definitions need some intuition. Have a look at Figure 3.1 to see that a wrap template simply stores some free time gaps (colored green in Figure 3.1) in a schedule where jobs may be placed. Remark that there can be at most one gap on each machine per definition. However, a wrap sequence is just a sequence of batches.

We use wrap templates to schedule wrap sequences in the following manner. Let $X = \bigcup_{l=1}^{k} C'_l$ be a set of jobs and job pieces of k different classes $i_1, \ldots, i_k \in [c]$ where $C'_l = \{j_1^l, \ldots, j_{n_l}^l\}$ is a set of jobs and job pieces of C_{i_l} for all $1 \le l \le k$. Furthermore, let $Q = [s_{i_l}, C'_l]_{l\in[k]}$ be a wrap sequence. Hence, Q has a length of $|Q| = \sum_{l=1}^{k}(1 + n_l) = k + \sum_{l=1}^{k} n_l$. Now let $\omega = ((u_1, a_1, b_1), \ldots, (u_{|\omega|}, a_{|\omega|}, b_{|\omega|}))$ be a wrap template. We want to schedule Q in McNaughton's wrap-around style using the gaps $[a_r, b_r]$ for $1 \le l \le |\omega|$. The critical point is when an item q hits the border b_r. If q is a setup, the solution is simple. In this case we simply move q below the next gap to be sure that the following jobs (or job pieces) get scheduled feasibly. If the critical item q is a job (piece) of a class i, we split q at time b_r into two new jobs (just like McNaughton's wrap-around rule) to place one job piece at the end of the current gap and the other job piece at the beginning of the next gap. As before, we add a setup s_i below the next gap to guarantee feasibility. We refer to the described algorithm as Wrap. Note that the critical job piece can be large such that it needs multiple gaps (and splits) to be placed. An algorithm for this splitting is given as Algorithm 1, namely Split. Split gets the current gap number r as well as the point in time t where q should be processed. It returns the gap number and the point in time where the next item should start processing. A simple comparison with McNaughton's wrap-around rule will prove the following lemma.

Lemma 7. *Let* Q *be a wrap sequence containing a largest setup* $s_{\max}^{(Q)}$ *and* ω *be a wrap template with* $\mathrm{L}(Q) \le S(\omega)$. *Then* Wrap *will place the last job (piece) of* Q *in a gap* ω_r *with* $r \le |\omega|$. *If there was a free time of at least* $s_{\max}^{(Q)}$ *below each gap but the first, the load gets placed feasibly.* □

Lemma 8. *If* $\mathrm{L}(Q) \le S(\omega)$ *then* Wrap(Q, ω) *has a running time of* $\mathcal{O}(|Q|+|\omega|)$.

Proof. Whenever a critical job (piece) is obtained, Wrap runs Split. Apparently each turn of the while loop in Split results in a switch to the next entry of ω, i. e. $r \leftarrow r + 1$, such that the total number of loop turns over all $q \in Q$ is bounded by $|\omega|$ thanks to Lemma 7. A turn of the for loop in Wrap needs constant time except for the execution of Split such that we get a total running time of $\mathcal{O}(|Q| + |\omega|)$. □

Algorithm 1 Split a critical job (piece) to the subsequent gaps and add necessary setups

procedure SPLIT(q, ω, r, t)
  Let i be the class of job (piece) q
  $p \leftarrow q$; $t' \leftarrow t + t_q$
  while $t' > b_r$ **do**
    Split q into new job pieces q_1, q_2
      with $t_{q_1} = t' - b_r$ and $t_{q_2} = b_r - t$
    Place job piece q_2 at time t on machine u_r
    $r \leftarrow r + 1$; $q \leftarrow q_1$; $t \leftarrow a_r$; $t' \leftarrow a_r + t_q$
                                ▷ turn to the next gap
    Place setup s_i at time $t - s_i$ on machine u_r
  done
  Place job piece q at time t on machine u_r
                                ▷ q $(= q_1)$ fits in the gap $[a_r, b_r]$
  return (r, t')

3.2.2 *Simple Upper Bounds*

For all three problems there is a 2-approximation with running time $\mathcal{O}(n)$. The algorithm for the splittable case is a simple application of wrap templates. For the non-preemptive and preemptive case one can use a slightly modified greedy solution.

Lemma 9. *There is a 2-approximation for the splittable case running in time $\mathcal{O}(n)$.*

Proof. Let I be an instance and let $T^{(1)}_{\min} = \max\{\frac{1}{m}N, s_{\max}\} \leq \mathrm{OPT}_{\mathrm{split}}(I)$ where $N = \sum_{i=1}^{c} s_i + \sum_{j \in J} t_j$. We construct a wrap template $\omega = (\omega_1, \ldots, \omega_m)$ of length $|\omega| = m$ by setting $\omega_r = (r, s_{\max}, s_{\max} + \frac{1}{m}N)$ for all $r \in [m]$, to wrap a wrap sequence $Q = [s_i, C_i]_{i \in [c]}$ containing all classes/jobs. Apparently we have $S(\omega) = \sum_{r=1}^{m} \frac{1}{m}N = N = \mathrm{L}(Q)$ and obviously the time $s_{\max}$ below each gap is sufficient to place the possibly missing setups. Hence, thanks to Lemma 7 this wrapping builds a feasible schedule with a makespan of at most $s_{\max} + \frac{1}{m}N \leq 2T^{(1)}_{\min} \leq 2\,\mathrm{OPT}_{\mathrm{split}}(I)$. The attentive reader recognizes that WRAP runs in time $\mathcal{O}(|Q| + |\omega|) = \mathcal{O}(c + n + m) > \mathcal{O}(n)$ (cf. Lemma 8) but actually a smarter implementation of SPLIT is able to overcome this issue. For details see the proof of Theorem 7 on page 49. □

Due to the same lower bounds (cf. Observations 2 and 3) the non-preemptive and preemptive case can be approximated using the same algorithm, stated in the proof of the following Lemma 10.

Lemma 10. *There is a 2-approximation for both the non-preemptive and preemptive case running in time $\mathcal{O}(n)$.*

Proof. Let I be an instance and let $T_{\min} = \max\{\frac{1}{m}N, \max_{i \in [c]}(s_i + t^{(i)}_{\max})\}$ where $t^{(i)}_{\max} = \max_{j \in C_i} t_j$ and $N = \sum_{i=1}^{c} s_i + \sum_{j \in J} t_j$. Consider the non-preemptive case and remark that $T_{\min} \leq \mathrm{OPT}_{\mathrm{nonp}}(I)$. First, we group the jobs by

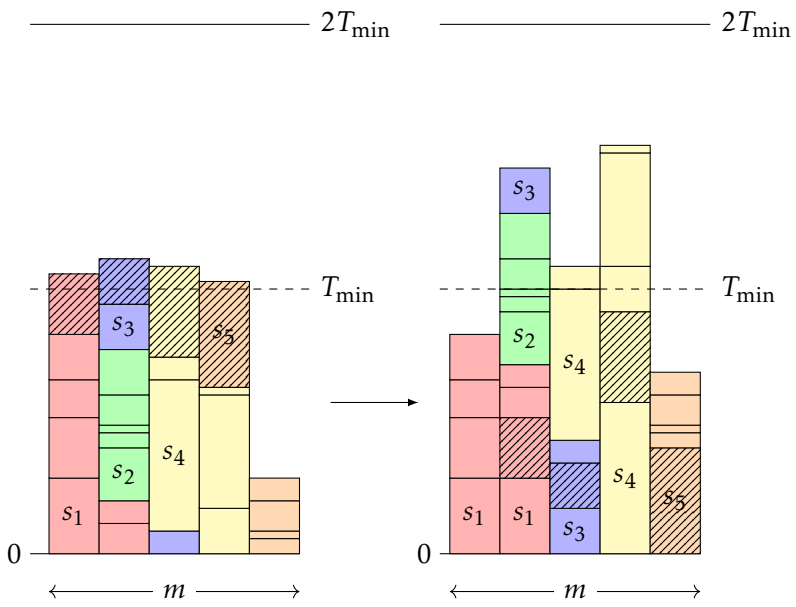


Figure 3.2: Example for a next-fit schedule with $m = c = 5$

classes. Let $C_i = \{j_1^i, \ldots, j_{n_i}^i\}$ with $n_i = |C_i|$ for all classes $i \in [c]$. Beginning on machine 1, we add one setup for each class followed by all jobs of the class. Whenever the load of the current machine exceeds $T_{\min}$, we keep the last placed item and proceed to the next machine. So we add the items $s_1, j_1^1, \ldots, j_{n_1}^1, s_2, j_1^2, \ldots, j_{n_2}^2, \ldots, s_c, j_1^c, \ldots, j_{n_c}^c$ to the machines using a next-fit strategy with threshold $T_{\min}$ (see Figure 3.2 on the left). The idea of the next step is to move the items (both jobs and setups) that cross the border $T_{\min}$ to the beginning of the next machine. For each moved item that was a job $j \in C_k$ we place an additional setup s_k right before j. All other load is shifted up as much as required to place the moved items (see Figure 3.2 on the right). In a last step one can remove unnecessary setups, i. e. setups which are scheduled last on a machine. In Figure 3.2 this reduces the load of machine 2 by the setup time s_3. Remark that the T-crossing items of the first step are hatched.

ANALYSIS First consider the first step. The load placed is exactly

$$\sum_{i=1}^{c}(s_i + P(C_i)) = \sum_{i=1}^{c} s_i + \sum_{j \in J} t_j = N = m \cdot \tfrac{1}{m}N \leq m \cdot T_{\min}$$

and this states that the load of the last machine is at most $T_{\min}$. Apparently the makespan of the resulting schedule is at most $T_{\min} + \max(s_{\max}, t_{\max})$. Now turn to the second step and consider a machine $u < m$. Passing away the item that exceeds $T_{\min}$, the load of u reduces to at most $T_{\min}$. Finally it increases to at most $T_{\min} + \max_{i \in [c]}(s_i + t_{\max}^{(i)}) \leq T_{\min} + T_{\min} = 2T_{\min}$ since u potentially receives an item q from machine $u - 1$ as well as an initial setup if q is a job. As mentioned above, the last machine $u = m$ already has a load of at most $T_{\min}$. So after the reassignment its load holds the bound of $2T_{\min}$ as well. Hence, in total the schedule has a makespan of at most

$2T_{\min} \leq 2\,\mathrm{OPT}$ and apparently both steps do run in time $\mathcal{O}(n)$ such that the described algorithm runs in time $\mathcal{O}(n)$.

Every solution to the non-preemptive case is a solution to the preemptive case and we obtained the same lower bounds for the preemptive case as for the non-preemptive one, i. e. $T_{\min} \leq \mathrm{OPT}_{\mathrm{pmtn}}(I)$. So the approximation can be used in the preemptive case as well. □

3.3 OVERVIEW

Here we give a briefly overview to our results. We start with our general results.

Theorem 4. *For all three problems there is a 2-approximation running in time* $\mathcal{O}(n)$. □

For the details see Section 3.2.2 on page 40. Especially if the reader is not familiar to these problems, the simple 2-approximations in Section 3.2.2 might be a good point to start.

We use the well-known approach of *dual approximation* (cf. Chapter 1) to get the following result.

Theorem 5. *For all three problems there is a* $(1.5+\varepsilon)$*-approximation running in time* $\mathcal{O}(n \log 1/\varepsilon)$.

Remark that already this result is much stronger for the preemptive case than the previous ratio of 2 by Monma and Potts. In more detail, we find 1.5-dual approximations for all three problem variants, all running in time $\mathcal{O}(n)$. Also in all problem cases there is a value $T_{\min}$ depending only on the input such that $\mathrm{OPT} \in [T_{\min}, 2T_{\min}]$ due to the 2-approximations. So a binary search suffices. In the following we briefly describe these dual approximations.

3.3.1 *Preemptive Scheduling*

Also in the setup context preemptive scheduling means that each job may be preempted at any time, *but* it is allowed to be processed on at most *one* machine at a time. In other words, a job may not run in parallel time. So, this is a job-constraint only; in fact, the load of a class may be processed in parallel but not the jobs themselves.

Observation 2. $\mathrm{OPT}_{\mathrm{pmtn}} \geq \max_{i\in[c]}(s_i + t^{(i)}_{\max})$ *where* $t^{(i)}_{\max} = \max_{j\in C_i} t_j$.

Proof. Let σ be a feasible schedule for an instance I with a makespan T and consider a job $j \in C_i$ of a class $i \in [c]$. There may be k job pieces $j_1, \ldots, j_k$ of job j with a total processing time of $\sum_{l=1}^{k} t_{j_l}$. Let p_l be the point in time σ starts to schedule job piece j_l. Without loss of generality we assume $p_1 \leq \cdots \leq p_k$. So the execution of job j ends at time $e_j = p_k + t_{j_k}$. Now remark that $p_l \leq p_{l+1}$ means that $p_l + t_{j_l} \leq p_{l+1}$ since otherwise j_l and j_{l+1} run in parallel time. It follows that $p_1 + \sum_{l=1}^{k-1} t_{j_l} \leq p_k$, which means $p_1 + t_j \leq p_k + t_{j_k} = e_j$. There is at least one setup s_i before time p_1, i. e. $p_1 \geq s_i$, and we obtain $T \geq e_j \geq p_1 + t_j \geq s_i + t_j$. □

Due to this, we assume that $m < n$ in the preemptive case, because $m \geq n$ leads to a trivial optimal solution by simply scheduling one job (and a setup) per machine.

The preemptive case appears to be very natural on the one hand but hard to approximate (for arbitrary large batches) on the other hand. Aiming for the ratio of 1.5, we managed to reduce the problem to a knapsack problem efficiently solvable as a *continuous* knapsack problem. Therefore, we need to take a closer look on I_{exp} and I_{chp} so we split them again. We divide the expensive classes into three disjoint subsets I^+_{exp}, I^0_{exp} and I^-_{exp} such that $i \in I_{\text{exp}}$ holds $i \in I^+_{\text{exp}}$ iff. $T \leq s_i + P(C_i)$, $i \in I^0_{\text{exp}}$ iff. $\frac{3}{4}T < s_i + P(C_i) < T$ and $i \in I^-_{\text{exp}}$ iff. $s_i + P(C_i) \leq \frac{3}{4}T$. Also we divide the cheap classes into I^+_{chp}, I^-_{chp} s.t. $i \in I_{\text{chp}}$ holds $i \in I^+_{\text{chp}}$ iff. $\frac{1}{4}T \leq s_i \leq \frac{1}{2}T$ and $i \in I^-_{\text{chp}}$ iff. $s_i < \frac{1}{4}T$.

Definition 8 (Nice Instances)**.** *For a makespan T we call an instance* nice *if I^0_{exp} is empty.*

The next theorem yields a 1.5-dual approximation for nice instances and will be important to find a 1.5-ratio for general instances too.

Theorem 10. *Let I be a nice instance for a makespan T. Moreover, let*

$$L_{\text{nice}} = P(J) + \sum_{i \in I^+_{\text{exp}}} \alpha'_i s_i + \sum_{i \in I^-_{\text{exp}} \cup I_{\text{chp}}} s_i$$

and $m_{\text{nice}} = \lceil \frac{1}{2}|I^-_{\text{exp}}| \rceil + \sum_{i \in I^+_{\text{exp}}} \alpha'_i$ where $\alpha'_i = \left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor$. Then the following properties hold.

(i) If $mT < L_{\text{nice}}$ or $m < m_{\text{nice}}$, it is true that $T < \text{OPT}_{\text{pmtn}}(I)$.

(ii) Otherwise a feasible schedule with makespan at most $\frac{3}{2}T$ can be computed in time $\mathcal{O}(n)$.

It turns out that nice instances are some sort of well-behaving instances which can be handled very easily *and* actually their definition is helpful for general instances too.

The motivation behind a general algorithm is the following. Obviously jobs of different expensive classes can not be placed on a common machine in a T*-feasible* schedule (a feasible schedule with a makespan of at most T). Especially the jobs of $J(I^0_{\text{exp}})$ and $J(I^+_{\text{exp}} \cup I^-_{\text{exp}})$ cannot. So we first place the classes of I^0_{exp} on one machine per class, which is reasonable as we will see later (cf. Lemma 15 on page 69). They obviously fit on a single machine, since $\frac{3}{4}T < s_i + P(C_i) < T$ for all $i \in I^0_{\text{exp}}$. Each of these *large* machines got free processing time less than $\frac{1}{4}T$ in a T-feasible schedule. After that we decide which jobs of cheap classes will get processing time on the large machines or get processed as part of a *nice* instance with the residual load that is scheduled on the residual $m - |I^0_{\text{exp}}|$ machines. Apparently only jobs of $I^-_{\text{chp}} \subseteq I_{\text{chp}}$ can actually be processed on large machines in a T-feasible schedule, because the setups of other cheap classes have a size of at least $\frac{1}{4}T$ so we only need to decide about this set. We will find a fine-grained knapsack instance on an appropriate subset for this decision. See Section 3.6 for the details.

3.3.2 *Splittable Scheduling*

In case of the splittable problem, jobs are allowed to be preempted at any time and all jobs (or job pieces) can be placed on any machine at any time. Especially jobs are allowed to be processed in parallel time (on different machines). It is important to notice that one should *not* assume $n \geq m$ in the splittable case, since increasing the number of machines may result in a lower (optimal) makespan; in fact, every optimal schedule makes use of *all* m machines. Due to this, it is remarkable that we allow a weaker definition of schedules in the following manner. A schedule may consist of machine configurations with associated multiplicities instead of (for example) explicitly mapping each job (piece) j to a pair $(u_j, x_j) \in [m] \times \mathbb{Q}$ where u_j is the machine on which j starts processing at time x_j.

Theorem 7. *Let I be an instance and let T be a makespan. Let*

$$L_{\text{split}} = P(J) + \sum_{i \in I_{\text{chp}}} s_i + \sum_{i \in I_{\text{exp}}} \beta_i s_i$$

and $m_{\text{exp}} = \sum_{i \in I_{\text{exp}}} \beta_i$. Then the following properties hold.

(i) If $mT < L_{\text{split}}$ or $m < m_{\text{exp}}$, then it is true that $T < \text{OPT}_{\text{split}}(I)$.

(ii) Otherwise a feasible schedule with makespan at most $\frac{3}{2}T$ can be computed in time $\mathcal{O}(n)$.

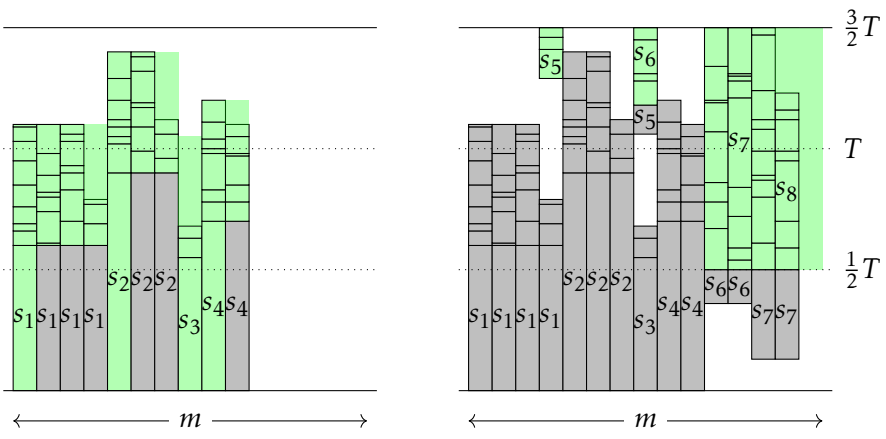
The idea of the algorithm is rather simple. We schedule the expensive classes by using as few setups as possible (imagining an optimal makespan, i. e. $T = \text{OPT}(I)$). An optimal schedule needs at least α_i setups/machines to schedule a class $i \in I_{\text{exp}}$, but we will only use $\beta_i \leq \alpha_i$ setups/machines (cf. Lemma 5). For each expensive class i we may get at most one machine $\bar{u}_i$ with a load $\text{L}(\bar{u}_i) < T$. So we can reserve the time interval of $[\text{L}(\bar{u}_i), \text{L}(\bar{u}_i) + \frac{1}{2}T]$ for a cheap setup on these machines before filling the residual time of $T - \text{L}(\bar{u}_i)$ with load of cheap classes, since $\text{L}(\bar{u}_i) + \frac{1}{2}T + (T - \text{L}(\bar{u}_i)) = \frac{3}{2}T$. Once all machines are filled up, we turn to unused machines and wrap between time $\frac{1}{2}T$ and $\frac{3}{2}T$ such that cheap setups can be placed below line $\frac{1}{2}T$. Figures 3.3a and 3.3b illustrate an example situation after step (1) and (2) with green colored wrap templates. See Section 3.4 on page 48 for the details.

3.3.3 *Non-Preemptive Scheduling*

Doing *non-preemptive* scheduling we do *not* allow jobs to be preempted. Even an optimal schedule needs to place at least one setup to schedule a job on a machine, so we find another lower bound.

Observation 3. $\text{OPT}_{\text{nonp}} \geq \max_{i \in [c]}(s_i + t^{(i)}_{\max})$ *where* $t^{(i)}_{\max} = \max_{j \in C_i} t_j$.

Therefore, analogous to preemptive scheduling we assume that $m < n$. Let $J_+ = \{ j \in J \mid t_j > \frac{1}{2}T \}$ be the big jobs whereas the small jobs be denoted by $J_- = \{ j \in J \mid t_j \leq \frac{1}{2}T \}$. Our algorithm is based on the fact, that there are



(a) Situation after step (1)

(b) Situation after step (2)

Figure 3.3: An example for the algorithm for the splittable case with $I_{\text{exp}} = \{1, 2, 3, 4\}$ and $I_{\text{chp}} = \{5, 6, 7, 8\}$

three subsets of jobs such that pairwise they cannot be scheduled on a single machine. These are J_+, $J(I_{\text{exp}})$, and $K = \bigcup_{i \in I_{\text{chp}}} \{j \in C_i \cap J_- \mid s_i + t_j > \frac{1}{2}T\}$.

In Section 3.5 we find the following minimum number of machines for each class. Let

$$m_i = \begin{cases} \left\lceil \frac{P(C_i)}{T - s_i} \right\rceil = \alpha_i & : i \in I_{\text{exp}} \\ |C_i \cap J_+| + \left\lceil \frac{P(C_i \cap K)}{T - s_i} \right\rceil & : i \in I_{\text{chp}} \end{cases}$$

for all $i \in [c]$. The following result yields our 1.5-dual approximation.

Theorem 9. *Let I be an instance and let T be a makespan. Let*

$$L_{\text{nonp}} = P(J) + \sum_{i=1}^{c} m_i s_i + \sum_{i : x_i > 0} s_i$$

and $m' = \sum_{i=1}^{c} m_i$ where $x_i = P(C_i) - m_i(T - s_i)$. Then the following properties hold.

(i) If $mT < L_{\text{nonp}}$ or $m < m'$, then it is true that $T < \text{OPT}_{\text{nonp}}(I)$.

(ii) Otherwise a feasible schedule with makespan at most $\frac{3}{2}T$ can be computed in time $\mathcal{O}(n)$.

Now binary search leads to a constant approximation:

Theorem 8. *There is a 1.5-approximation for the non-preemptive case running in time $\mathcal{O}(n \log(n + \Delta))$ where $\Delta = \max\{s_{\max}, t_{\max}\}$ is the largest number of the input.*

Proof. Unlike the other cases, here the optimal value is an integral number, i. e. $\text{OPT}_{\text{nonp}} \in \mathbb{Z}_{\geq 1}$, since all values in the input are integral numbers and neither jobs nor setups are allowed to be preempted. Therefore, a binary search in $[T_{\min}, 2T_{\min}]$ can find an appropriate makespan in time $\lceil \log T_{\min} \rceil \cdot$

$\mathcal{O}(n) = \mathcal{O}(n \log T_{\min})$ and it is easy to show that $\log T_{\min} \leq \mathcal{O}(\log(n + \Delta))$. Given the 1.5-dual approximation of Theorem 9 this completes the proof of Theorem 8. □

See Section 3.5 on page 51 for all the details.

3.3.4 *Class Jumping*

With a different idea for a binary search routine for an appropriate makespan we are able to improve both the running time and the approximation ratio for the splittable and preemptive case. As with a single binary search we test makespan guesses with our dual algorithms. The general idea is to look at some points in time which we call *jumps*. A *jump* of an expensive class i is some makespan guess T such that any lower guess $T' < T$ will cause at least one more setup/machine to schedule the jobs of class i. The goal is to find two jumps $T_{\text{fail}}, T_{\text{ok}}$ of two classes such that there is no jump of any other class between them, while T_{fail} is rejected and T_{ok} is accepted. In fact, this means that any makespan T between both jumps causes the same load L (with our dual algorithm). Therefore, either T_{ok} or $\frac{1}{m}L$ will be an appropriate makespan.

Theorem 6. *There is a* 1.5*-approximation for the splittable case running in time* $\mathcal{O}(n + c \log(c + m))$.

With a small modification the idea can be reused to be applied to the preemptive case as well and this yields our strongest result for the preemptive case.

Theorem 12. *There is a* 1.5*-approximation for the preemptive case running in time* $\mathcal{O}(n \log n)$.

Here we only present the improvement for the splittable case. The improved search for the preemptive case is slightly more complicated and based even more on the details of the 1.5-dual approximation so we refer to Section 3.6.4 for the details.

The following ideas are crucial. Once the total processing times $P_i = P(C_i)$ are computed, the values β_i can be computed in constant time and one can test Theorem 7(ii) in time $\mathcal{O}(c)$ for a given makespan T. Whenever we test a makespan T we save the computed values β_i as $\beta_i(T)$. We will call $(A, B]$ a *right interval* if makespan B satisfies Theorem 7(ii) (B is *accepted*) while A does not (A is *rejected*). For example $(s_{\max}, N]$ is a right interval.

We go through the interesting details of Algorithm 2 to get the idea of the search routine.

Step 4. Note that $T \in [2\tilde{s}_{i-1}, 2\tilde{s}_i)$ means $\tilde{s}_0 \leq \cdots \leq \tilde{s}_{i-2} \leq \tilde{s}_{i-1} \leq \frac{1}{2}T < \tilde{s}_i \leq \tilde{s}_{i+1} \leq \cdots \leq \tilde{s}_{c+1}$ and so any makespan in an interval $[2\tilde{s}_{i-1}, 2\tilde{s}_i)$ causes the *same* partition $I_{\text{exp}} \dot{\cup} I_{\text{chp}}$. The running time can be obtained with binary search.

Step 5. We call T a *jump* of a class $i \in I_{\text{exp}}$ if $2P_i/T$ is integer. That means all machines containing jobs of class i are filled up to line $s_i + \frac{1}{2}T$. So T represents a point in time such that any $T' < T$ will cause at least one more

Algorithm 2 Class Jumping for Splittable Scheduling

1. Set $\tilde{s}_0 = 0$ and $\tilde{s}_{c+1} = N$
2. Sort the setup time values s_i ascending and name them $\tilde{s}_1, \dots, \tilde{s}_c$ in time $\mathcal{O}(c \log c)$
3. Compute $P_i = P(C_i)$ for all $i \in [c]$ in time $\mathcal{O}(n)$
4. Guess the right makespan interval $V = (2\tilde{s}_{i-1}, 2\tilde{s}_i]$ in time $\mathcal{O}(c \log c)$ and set $T_1 = 2\tilde{s}_i$
5. Find a *fastest jumping class* $f \in I_{\text{exp}}$, i. e. $P_f \geq P_i$ for all $i \in I_{\text{exp}}$ in time $\mathcal{O}(c)$
6. Guess the right interval $W = (2P_f/(\beta_f(T_1) + k + 1), 2P_f/(\beta_f(T_1) + k)]$ for some integer $k < m$ such that $X := V \cap W \neq \emptyset$, in time $\mathcal{O}(c \log m)$ and set $T_2 = 2P_f/(\beta_f(T_1) + k)$
7. Find and sort the $\mathcal{O}(c)$ jumps in X in time $\mathcal{O}(c \log c)$
8. Guess the right interval $Y = (T_{\text{fail}}, T_{\text{ok}}] \subseteq W$ for two jumps $T_{\text{fail}}, T_{\text{ok}}$ of two classes $i_a, i_b \in I^+_{\text{exp}}$ which jump in X such that no *other* class jumps in Y, in time $\mathcal{O}(c \log c)$
9. Choose a suitable makespan in this interval in constant time

(obligatory) setup to schedule class i. It takes a time of $\mathcal{O}(c)$ to find some class $f \in I_{\text{exp}}$ with $P_f = \max_{i \in I_{\text{exp}}} P_i$.

Step 6. Just remark that $2P_f/(\beta_f(T_1) + k)$ and $2P_f/(\beta_f(T_1) + k + 1)$ are two consecutive jumps of class f.

Step 7. In the analysis we will see that X contains at most c jumps in total (of all classes). To find a jump of a class $\iota \in I_{\text{exp}}$ in X just look at $\beta_\iota(T_2)$. If $2P_\iota/\beta_\iota(T_2) < X$ then there is no jump of class ι in X. Otherwise $T_\iota := 2P_\iota/\beta_\iota(T_2)$ is the only jump of class ι in X.

Step 9. So T_{ok} was accepted while T_{fail} got rejected and there are no jumps of any other classes between them. Let $L_{\text{split}}(T_{\text{fail}})$ be the load which is required to place T_{fail} and set $T_{\text{new}} = \frac{1}{m} L_{\text{split}}(T_{\text{fail}})$.

Case $m < m_{\text{exp}}(T_{\text{fail}})$. So the jump causes too many required machines and hence $T < T_{\text{ok}}$ means $T < \text{OPT}$. We return T_{ok}.

Case $m \geq m_{\text{exp}}(T_{\text{fail}})$. We do another case distinction as follows.

If $T_{\text{new}} \geq T_{\text{ok}}$ we find that T_{ok} is smaller than the smallest makespan that may be suitable to place $L_{\text{split}}(T_{\text{fail}})$. Therefore, we return T_{ok}.

If $T_{\text{new}} < T_{\text{ok}}$ it follows $T_{\text{fail}} = \frac{mT_{\text{fail}}}{m} < \frac{1}{m} L_{\text{split}}(T_{\text{fail}}) = T_{\text{new}} < T_{\text{ok}}$ since $m \geq m_{\text{exp}}(T_{\text{fail}})$ and the rejection of T_{fail} implies that $mT_{\text{fail}} < L_{\text{split}}(T_{\text{fail}})$. So we have $T_{\text{new}} \in (T_{\text{fail}}, T_{\text{ok}})$ and we get $m \geq m_{\text{exp}}(T_{\text{fail}}) = m_{\text{exp}}(T_{\text{new}})$ and $mT_{\text{new}} = L_{\text{split}}(T_{\text{fail}}) = L_{\text{split}}(T_{\text{new}})$. Because of Theorem 7 we return T_{new}.

The interesting part of the analysis is the fact, that there are no more than $\mathcal{O}(c)$ jumps in X and we want to show the following lemma.

Lemma 11. *If T' is a jump of f, i. e. $T' = 2P_f/\beta_f(T')$, and T'' is a jump of a different class i, i. e. $T'' = 2P_i/\beta_i(T'')$, such that $T'' \leq T'$, then the next jump of class i is smaller than the next jump of f, which can be written as*

$$\frac{2P_i}{\beta_i(T'')+1} \leq \frac{2P_f}{\beta_f(T')+1}.$$

Proof. Since $T'' \leq T'$ we have

$$\beta_i(T'')P_f = \frac{2P_i}{T''}P_f \geq \frac{2P_i}{T'}P_f = \beta_f(T')P_i \tag{3.1}$$

and thus it follows that

$$\frac{2P_i}{\beta_i(T'')+1} = \frac{2P_fP_i}{\beta_i(T'')P_f+P_f} \overset{(3.1)}{\leq} \frac{2P_fP_i}{\beta_f(T')P_i+P_f} \overset{P_f \geq P_i}{\leq} \frac{2P_fP_i}{\beta_f(T')P_i+P_i} = \frac{2P_f}{\beta_f(T')+1}.$$

□

Proof of Theorem 6. Due to Lemma 11 any class that jumps in X jumps *outside* of X the next time. So for every class $i \in I_{\exp}$ there is at most one jump in X and hence X contains at most $|I_{\exp}| \leq c$ jumps. Apparently the sum of the running times of each step is $\mathcal{O}(n + c\log(c+m))$ and the returned value $T \in \{T_{\text{ok}}, T_{\text{new}}\}$ holds $T \leq \text{OPT}_{\text{split}}$ while being accepted by Theorem 7(ii) such that the algorithm for the splittable case computes a feasible schedule with ratio 3/2 in time $\mathcal{O}(n)$. The total running time is $\mathcal{O}(n+c\log(c+m))$. □

3.4 SPLITTABLE SCHEDULING

For this section let $T_{\min} = \max\{\frac{1}{m}N, s_{\max}\}$ where $N = \sum_{i=1}^{c} s_i + \sum_{j\in J} t_j$. Let I be an instance and let $T \geq T_{\min}$ be a makespan. We describe the algorithm in two steps.

STEP 1 First we place all jobs of expensive classes. We define a wrap template $\omega^{(i)}$ of length $|\omega^{(i)}| = \lceil 2P(C_i)/T \rceil = \beta_i$ for each class $i \in I_{\exp}$ as follows. Let $\omega_1^{(i)} = (u_i, 0, s_i + \frac{1}{2}T)$ and $\omega_{1+r}^{(i)} = (u_i + r, s_i, s_i + \frac{1}{2}T)$ for $1 \leq r < \beta_i$. Here the first machines u_i have to be chosen distinct to all machines of the other wrap templates. We convert the expensive classes $i \in I_{\exp}$ into simple wrap sequences $Q^{(i)} = [s_i, C_i]$ consisting of an initial setup s_i followed by an arbitrary order of all jobs in C_i. For all $i \in I_{\exp}$ we use $\text{WRAP}(Q^{(i)}, \omega^{(i)})$ to wrap $Q^{(i)}$ into $\omega^{(i)} = (\omega_1^{(i)}, \ldots, \omega_{\beta_i}^{(i)})$. Remark that WRAP places a setup s_i at time 0 on each machine $u_i + l$ where $1 \leq l < \beta_i$. See Figure 3.3a for an example.

STEP 2 The next and last step is to place the jobs of cheap classes. Let $\bar{u}_i$ be the *last* machine used to wrap a sequence $Q^{(i)}$ in the previous step, i. e. $\bar{u}_i = u_i + \beta_i - 1$ for all $i \in I_{\exp}$. The idea is to use the free time left on the

machines $\bar{u}_i$ while reserving a time of exactly $\frac{1}{2}T$ for a cheap setup. Once these machines are filled, we turn to unused machines. In more detail we define one wrap template ω and one wrap sequence Q to place all jobs $J(I_{\text{chp}})$ as follows. Let $m_{\text{exp}} = \sum_{i \in I_{\text{exp}}} \beta_i$ be the number of machines used in step (1) and let $i_1, \dots, i_p \in I_{\text{exp}}$ be all p classes $i \in I_{\text{exp}}$ that hold $\mathrm{L}(\bar{u}_i) < T$. We define $\omega_l = (\bar{u}_{i_l}, \mathrm{L}(\bar{u}_{i_l}) + \frac{1}{2}T, \frac{3}{2}T)$ for all $1 \leq l \leq p$. To fill the residual (and empty) $k = m - m_{\text{exp}}$ machines $r_1, \dots, r_k \in [m]$ we set $\omega_{p+l} = (r_l, \frac{1}{2}T, \frac{3}{2}T)$ for all $1 \leq l \leq k$. The wrap sequence $Q = [s_i, C_i]_{i \in I_{\text{chp}}}$ simply consists of all jobs of $J(I_{\text{chp}}) = \bigcup_{i \in I_{\text{chp}}} C_i$ with an initial setup s_i before all the jobs of C_i for a cheap class $i \in I_{\text{chp}}$. As predicted, we wrap Q into $\omega = (\omega_1, \dots, \omega_p, \omega_{p+1}, \dots, \omega_{p+m-m_{\text{exp}}})$ using $\textsc{Wrap}(Q, \omega)$. See Figure 3.3b for an example.

3.4.1 *Analysis*

We want to show the following lemma.

Theorem 7. *Let I be an instance and let T be a makespan. Let*

$$L_{\text{split}} = P(J) + \sum_{i \in I_{\text{chp}}} s_i + \sum_{i \in I_{\text{exp}}} \beta_i s_i$$

and $m_{\text{exp}} = \sum_{i \in I_{\text{exp}}} \beta_i$. Then the following properties hold.

(i) If $mT < L_{\text{split}}$ or $m < m_{\text{exp}}$, then it is true that $T < \mathrm{OPT}_{\text{split}}(I)$.

(ii) Otherwise a feasible schedule with makespan at most $\frac{3}{2}T$ can be computed in time $\mathcal{O}(n)$.

Proof. (i). We show that $T \geq \mathrm{OPT}_{\text{split}}$ implies $mT \geq L_{\text{split}}$ and $m \geq m_{\text{exp}}$. Let $T \geq \mathrm{OPT}_{\text{split}}$. Then there is a feasible schedule σ with makespan T and $\mathrm{L}(\sigma) = \sum_{i=1}^{c} (\lambda_i^{(\sigma)} s_i + P(C_i))$. Due to Lemma 5, we have

$$\begin{aligned} mT \geq \mathrm{L}(\sigma) &= \sum_{i=1}^{c} (\lambda_i^{\sigma} s_i + P(C_i)) \\ &\geq P(J) + \sum_{i=1}^{c} \alpha_i s_i \geq P(J) + \sum_{i \in I_{\text{chp}}} s_i + \sum_{i \in I_{\text{exp}}} \beta_i s_i = L_{\text{split}}. \end{aligned}$$

Also $m \geq m_{\text{exp}}$ is a direct consequence of Lemmas 5 and 6.

(ii). Let $mT \geq L_{\text{split}}$ and $m \geq m_{\text{exp}}$. Note that the number of machines used in step (1) is $\sum_{i \in I_{\text{exp}}} \beta_i = m_{\text{exp}} \leq m$ and hence we have enough machines but we have to check for all $i \in I_{\text{exp}}$ that the wrap template $\omega^{(i)}$ in step (1) is suitable to wrap $Q^{(i)} = [s_i, C_i]$ into it, i. e. $S(\omega^{(i)}) \geq \mathrm{L}(Q^{(i)})$. This is true since

$$S(\omega^{(i)}) = s_i + \beta_i \cdot \tfrac{1}{2}T = s_i + \left\lceil \frac{P(C_i)}{\frac{1}{2}T} \right\rceil \cdot \tfrac{1}{2}T \geq s_i + P(C_i) = \mathrm{L}(Q^{(i)}).$$

Each wrap template $\omega^{(i)}$ is filled with exactly one class i and reserves a time s_i below each gap. So there is enough space to place a setup s_i below all gaps

of $\omega^{(i)}$. It remains to show that the wrap template ω in step (2) is suitable to wrap Q into it. This needs a bit more effort. Apparently for all $i \in I_{\text{exp}}$ the load $\mathrm{L}(\bar{u}_i)$ of the last machine $\bar{u}_i = u_i + \beta_i - 1$ holds

$$\beta_i s_i + P(C_i) = (\beta_i - 1)(s_i + \tfrac{1}{2}T) + \mathrm{L}(\bar{u}_i) \geq (\beta_i - 1)T + \mathrm{L}(\bar{u}_i)$$

since $s_i \geq \frac{1}{2}T$. Hence, if the last machine is filled to at least T, i. e. $\mathrm{L}(\bar{u}_i) \geq T$, we obtain that $\beta_i s_i + P(C_i) \geq \beta_i T$ and otherwise it follows that $\beta_i s_i + P(C_i) + T - \mathrm{L}(\bar{u}_i) \geq \beta_i T$. These two inequalities imply that

$$\begin{aligned} L' &:= \sum_{\substack{i \in I_{\text{exp}} \\ \mathrm{L}(\bar{u}_i) < T}} (T - \mathrm{L}(\bar{u}_i)) + \sum_{i \in I_{\text{exp}}} (\beta_i s_i + P(C_i)) \\ &= \sum_{\substack{i \in I_{\text{exp}} \\ \mathrm{L}(\bar{u}_i) < T}} (\beta_i s_i + P(C_i) + T - \mathrm{L}(\bar{u}_i)) + \sum_{\substack{i \in I_{\text{exp}} \\ \mathrm{L}(\bar{u}_i) \geq T}} (\beta_i s_i + P(C_i)) \\ &\geq \sum_{\substack{i \in I_{\text{exp}} \\ \mathrm{L}(\bar{u}_i) < T}} \beta_i T + \sum_{\substack{i \in I_{\text{exp}} \\ \mathrm{L}(\bar{u}_i) \geq T}} \beta_i T = m_{\text{exp}} T \end{aligned}$$

and we use this inequality to show that ω is suitable to wrap Q since

$$\begin{aligned} S(\omega) &= \sum_{\substack{i \in I_{\text{exp}} \\ \mathrm{L}(\bar{u}_i) < T}} (T - \mathrm{L}(\bar{u}_i)) + (m - m_{\text{exp}})T \\ &\geq \sum_{\substack{i \in I_{\text{exp}} \\ \mathrm{L}(\bar{u}_i) < T}} (T - \mathrm{L}(\bar{u}_i)) + L_{\text{split}} - m_{\text{exp}} T && mT \geq L_{\text{split}} \\ &= L' + \sum_{i \in I_{\text{chp}}} (s_i + P(C_i)) - m_{\text{exp}} T \;\geq\; \mathrm{L}(Q). && L' \geq m_{\text{exp}} T \end{aligned}$$

One can easily confirm that the reserved processing time of $\frac{1}{2}T$ below all used gaps is sufficient. In detail, we only place jobs of cheap classes in step (2). So the call of $\textsc{Wrap}(Q, \omega)$ needs a time of at most $\frac{1}{2}T$ to place cheap setups below the gaps. Hence, the computed schedule is feasible and this proves Theorem 7.

Nevertheless we still need to analyze the running time. Apparently the running time directly depends on the running time of WrapSplit as follows. For step (1) we get a running time of $\mathcal{O}(1) + \sum_{i \in I_{\text{exp}}} \mathcal{O}(|Q^{(i)}| + |\omega^{(i)}|) = \sum_{i \in I_{\text{exp}}} \mathcal{O}(|C_i| + \beta_i) = \mathcal{O}(|J(I_{\text{exp}})| + \sum_{i \in I_{\text{exp}}} \beta_i) \leq \mathcal{O}(n+m)$ due to Lemma 8 since $|Q_i| = 1 + |C_i|$ and $|\omega_i| = \beta_i$ for all $i \in I_{\text{exp}}$. To optimize the running time we find another implementation[1] of Split (and Wrap) for our use case. As mentioned before we allow that a schedule consists of machine configurations with given multiplicities. In fact, there is a more efficient implementation of Split for ranges of wrap sequences where all gaps start and end at equal times, i. e. $a_{r_1} = a_{r_2}$ and $b_{r_1} = b_{r_2}$. Apparently Split will place at most three different gap types (or gap configurations) for each job (piece) in such ranges

1 A similar idea was already mentioned by Jansen et al. in [67]

of *parallel* gaps. To see that let $0 \leq a < b$ describe the gaps and consider a job (piece) j which we start to place at time $t \in [a, b)$. If j is split by SPLIT at most once, we obviously have at most two used gaps; hence, we have at most two different gap configurations. If j is split at least two times j is split into a first piece with processing time $b - t$ followed by $\mu_j := \lfloor (t_j - (b-t))/(b-a) \rfloor$ gaps filled with processing time $b - a$ and a last gap starting with a piece of time $t_j - (b-t) - \mu_j(b-a)$. These define at most three different gap configurations. Since the multiplicity μ_j of the in between gaps can be computed in constant time, we can compute these three gap configurations and its multiplicities in constant time. So we get a running time of $\mathcal{O}(n+c) = \mathcal{O}(n)$ for step (1). For step (2) we apply this technique only for the $m - m_{\text{exp}}$ last gaps $\omega_{p+1}, \ldots, \omega_{p-m-m_{\text{exp}}}$ which are parallel in our sense. Remark that $p \leq c$ to see that the running time is $\mathcal{O}(c + |Q|) \leq \mathcal{O}(n)$. Hence, we get a total running time of $\mathcal{O}(n)$. □

3.5 NON-PREEMPTIVE SCHEDULING

Doing *non-preemptive* scheduling we do *not* allow jobs to be preempted. Even an optimal schedule needs to place at least one setup to schedule a job on a machine, so remember Observation 3 which says $\text{OPT}_{\text{nonp}} \geq \max_{i \in [c]}(s_i + t^{(i)}_{\max})$ where $t^{(i)}_{\max} = \max_{j \in C_i} t_j$.
Analogous to preemptive scheduling we assume $m < n$. For this section let $T_{\min} = \max\{\frac{1}{m}N, \max_{i \in [c]}(s_i + t^{(i)}_{\max})\}$ where $N = \sum_{i=1}^{c} s_i + \sum_{j \in J} t_j$ and $t^{(i)}_{\max} = \max_{j \in C_i} t_j$.

Theorem 8. *There is a* 1.5*-approximation for the non-preemptive case running in time* $\mathcal{O}(n \log(n + \Delta))$ *where* $\Delta = \max\{s_{\max}, t_{\max}\}$ *is the largest number of the input.*

Let I be an instance and $T \geq T_{\min}$ be a makspan. For later purposes we split the jobs into big and small ones. In more detail, let $J_+ = \{ j \in J \mid t_j > \frac{1}{2}T \}$ and $J_- = \{ j \in J \mid t_j \leq \frac{1}{2}T \}$. In the following we will look at three subsets of J. They are J_+, $J(I_{\text{exp}}) = \bigcup_{i \in I_{\text{exp}}} C_i$ as well as $K := \bigcup_{i \in I_{\text{chp}}} \{ j \in C_i \cap J_- \mid s_i + t_j > \frac{1}{2}T \}$ and one can easily see that they are in pairs disjoint. Let $L = J_+ \dot\cup J(I_{\text{exp}}) \dot\cup K$.

Observation 4. *It is true that* $L = \bigcup_{i \in [c]} \{ j \in C_i \mid s_i + t_j > \frac{1}{2}T \}$. □

We find the following minimum number of machines for each class. For all $i \in [c]$ let

$$m_i = \begin{cases} \left\lceil \frac{P(C_i)}{T - s_i} \right\rceil = \alpha_i & : i \in I_{\text{exp}} \\ |C_i \cap J_+| + \left\lceil \frac{P(C_i \cap K)}{T - s_i} \right\rceil & : i \in I_{\text{chp}} \end{cases}.$$

Observation 5. *Different jobs in* L *of different classes have to be scheduled on different machines. Furthermore, every job in* $J_+ \subseteq L$ *needs an own machine.*

Proof. Assume that two jobs $j_1, j_2 \in L$ of different classes $i_1, i_2 \in [c]$ are scheduled feasibly on one machine u, i. e. $\text{L}(u) \leq T$. Due to Observation 4, we have $s_{i_1} + t_{j_1} > \frac{1}{2}T$ as well as $s_{i_2} + t_{j_2} > \frac{1}{2}T$. To schedule j_1 and j_2 on u

it needs at least one setup time for both of them since $i_1 \neq i_2$. So we get a total load of $\mathrm{L}(u) \geq (s_{i_1} + t_{j_1}) + (s_{i_2} + t_{j_2}) > \frac{1}{2}T + \frac{1}{2}T = T$, a contradiction. Now assume that $j_1, j_2 \in J_+$ are jobs of a common class $i \in [c]$ and they are scheduled feasibly on one machine u, i. e. $\mathrm{L}(u) \leq T$. Since both jobs are in the same class i we need only one setup time s_i, but the total load is $\mathrm{L}(u) \geq s_i + t_{j_1} + t_{j_2} > s_i + T > T$ since $t_{j_1} > \frac{1}{2}T$ and $t_{j_2} > \frac{1}{2}T$. Again, this is a contradiction. □

Lemma 12. *Let σ be a feasible schedule with makespan T. Then σ needs at least m_i different machines to schedule a class $i \in [c]$ and in total σ needs at least $\sum_{i=1}^{c} m_i$ different machines.* □

Lemma 12 is a consequence of Lemmas 5 and 6 and Observation 5. We look

Algorithm 3 A 1.5-dual Approximation for Non-Preemptive Scheduling

1. Schedule all jobs of $L = \bigcup_{i \in [c]} \{ j \in C_i \mid s_i + t_j > \frac{1}{2}T \}$ on m_i machines for each class i
2. Consider the already used machines and schedule as many jobs as possible of $J \setminus L = \bigcup_{i \in [c]} \{ j \in C_i \mid s_i + t_j \leq \frac{1}{2}T \}$ without adding new setup times
3. Take one new setup time for each remaining class and place the remaining jobs greedily
4. Make the schedule non-preemptive and add setup times as needed

at Algorithm 3 in more detail.

Step 1. We schedule the jobs of L. For every class $i \in [c]$ do the following. If i is expensive, we place all jobs of C_i preemptively (until T) with one initial setup time s_i at the beginning of each of the required machines. In detail, we use a wrap template $\omega^{(i)} = (\omega_1^{(i)}, \ldots, \omega_{\alpha_i}^{(i)})$ of length $|\omega(i)| = \alpha_i = \lceil P(C_i)/(T - s_i) \rceil$ with $\omega_1^{(i)} = (u_i, 0, T)$ and $\omega_{1+r}^{(i)} = (u_i + r, s_i, T)$ for a first machine u_i such that all used machines are distinct and $1 \leq r < \alpha_i$. We use this wrap template to schedule a simple wrap sequence $Q^{(i)} = [s_i, C_i]$ with $\textsc{Wrap}(Q^{(i)}, \omega^{(i)})$. If i is cheap, we place all the jobs $j \in C_i \cap J_+$ with an initial setup time s_i on a single unused machine $v_k^{(i)}$, i. e. the load of such a machine will be $\frac{1}{2}T < s_i + t_j \leq T$. After that we place all jobs of $C_i \cap K$ preemptively (until T) on unused machines with one initial setup time s_i at the beginning of each of the required machines. As before, we use a simple wrap template $\omega^{(i)} = (\omega_1^{(i)}, \ldots, \omega_{\alpha_i}^{(i)})$ of length $|\omega^{(i)}| = \alpha_i$ with $\omega_1^{(i)} = (u_i, 0, T)$ and $\omega_{1+r}^{(i)} = (u_i + r, s_i, T)$ for a first machine u_i such that all used machines are distinct and $1 \leq r < \alpha_i$. We use $\omega^{(i)}$ to wrap a wrap sequence $Q^{(i)} = [s_i, C_i \cap K]$ with $\textsc{Wrap}(Q^{(i)}, \omega^{(i)})$. For all classes $i \in [c]$ let $\bar{u}_i = u_i + m_i - 1$ be the last machine used to wrap the sequence $Q^{(i)}$. For an example schedule after this step see Figure 3.4. The dashed lines indicate the wrap templates and the wrap sequences are filled green (dark if preempted).

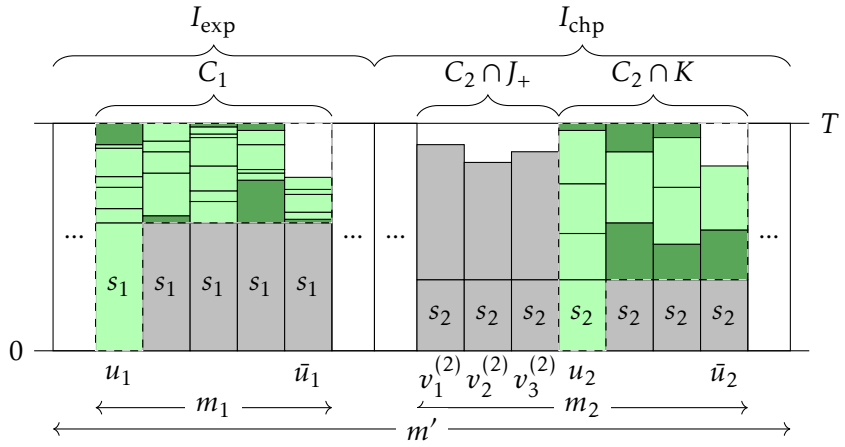


Figure 3.4: An example situation after step 1. of Algorithm 3 with $1 \in I_{\text{exp}}$ and $2 \in I_{\text{chp}}$

Step 2. Now we will place as many jobs as possible of $J \setminus L$ without adding new machines or setup times. Note that there is at most one setup time on a used machine so far. Let $v_1^{(i)}, \ldots, v_{k_i}^{(i)}$ be the machines used to schedule $C_i \cap J_+$ in step 1., i. e. $k_i = |C_i \cap J_+|$, and let $v_{k_i+1}^{(i)} = \bar{u}_i$. For every cheap class $i \in I_{\text{chp}}$ set $C_i' \leftarrow C_i \setminus L$ and start the following loop. Let $j \in C_i'$ and find a used machine $u = v_k^{(i)}$ for $1 \leq k \leq k_i + 1$ that has a load $L(u) < T$. If such a machine can not be found, the remaining jobs C_i' will be placed in step 3.. If $L(u) + t_j \leq T$ place j on top of machine u and set $C_i' \leftarrow C_i' \setminus \{j\}$. Otherwise split j into two new job pieces j_1, j_2 (of class i) such that $t_{j_1} = T - L(u)$ as well as $t_{j_2} = t_j - t_{j_1}$ and place j_1 on top of machine u and set $C_i' \leftarrow (C_i' \setminus \{j\}) \cup \{j_2\}$. Furthermore, we save j as the *parent job* of the new job pieces j_1 and j_2, i. e. we set $\text{parent}(j_1) \leftarrow j$ as well as $\text{parent}(j_2) \leftarrow j$. Process j_2 next in the loop. See Figure 3.5 for an example of this step. The new placed jobs of the cheap class $2 \in I_{\text{chp}}$ are colored blue (and dark if preempted). Remark that jobs may be split more than once; in fact, one can see that there is a job of class 2 that is split onto the machines $v_2^{(2)}$, $v_3^{(2)}$ and $v_4^{(2)} = \bar{u}_2$.

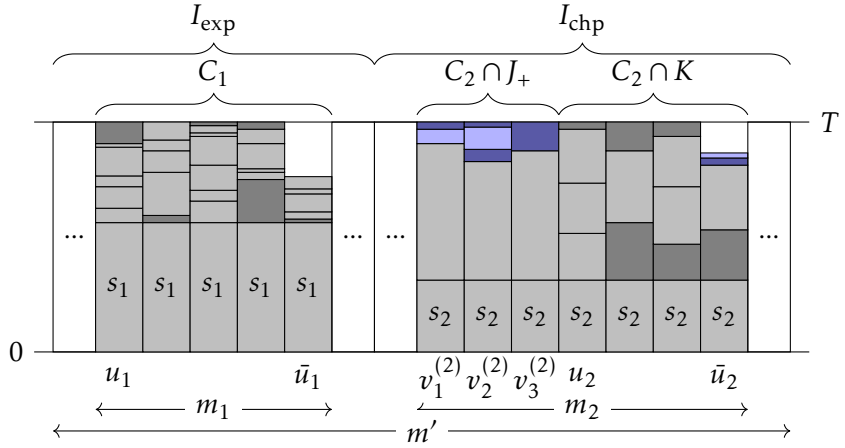


Figure 3.5: The situation after step 2. of Algorithm 3 with $1 \in I_{\text{exp}}$ and $2 \in I_{\text{chp}}$

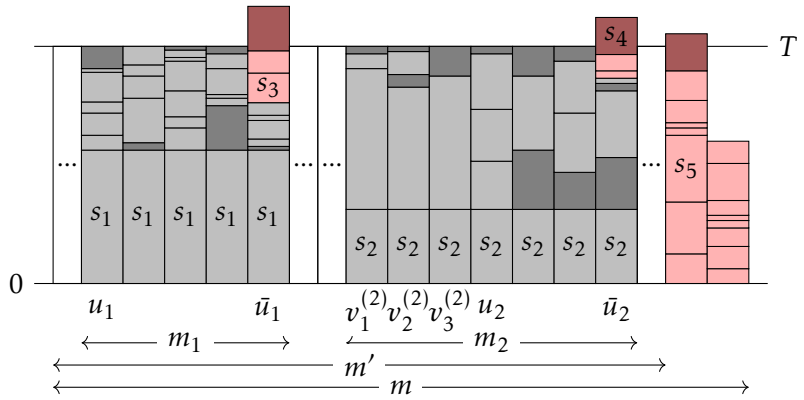


Figure 3.6: The situation after step 3. of Algorithm 3 with $1 \in I_{\text{exp}}$ and $\{2, 3, 4, 5\} \subseteq I_{\text{chp}}$

Step 3. Now we cannot schedule a job of C_i' for any $i \in I_{\text{chp}}$ without paying a new setup time s_i. However, we can discard classes i without residual load, i. e. $P(C_i') = 0$. So we build a wrap sequence $Q = [s_i, C_i']_{i:P(C_i')>0}$ that only contains classes with non-empty residual load. Instead of wrapping Q using a wrap template, we greedily fill up the used machines with a load *less* than T until an item crosses the border T. We do not split these critical items but just keep them as they are (*non-preempted*) and turn to the next machine. Once all used machines are filled to at least T, we fill up the unused machines in just the same manner. In Figure 3.6 one can see an example situation after this step where the items of Q are colored red (dark if T-crossing).

Step 4. The former solution is not feasible yet. That is due to a number of preemptively scheduled jobs on the one hand and the lack of some setup times on the other hand. The first step to obtain a non-preemptive solution is to consider each last job j on a machine. If j was scheduled integral, we keep it that way. If on the other hand j is the first part of a split of step 1. or step 2., we remove j from the machine and schedule the *parent job* $\text{parent}(j)$ instead. Also, we remove all other split pieces j' with $\text{parent}(j') = \text{parent}(j)$ from the schedule and shift down the above jobs by $t_{j'}$. Note that *all* jobs are placed *non-preemptively* now. The second step is to look upon the items scheduled in 3. *in the order they were placed*. Every item q that exceeds T in the current schedule (and therefore is last on its machine) is moved to the machine of item q' that was placed next. More precisely, q' and all jobs above q' are shifted up by $s_i + t_q$ if q is a job of class i or by $q = s_i$ if q is a setup. Accordingly s_i followed by q is placed at the free place below q' if q is a job of class i or $q = s_i$ is placed at the free place below q' if q is a setup. In the analysis we will see that this builds a feasible schedule with makespan at most $\frac{3}{2}T$. Have a look at Figure 3.7 to see an example result of Algorithm 3. All previously preempted or T-crossing items are colored dark.

3.5.1 *Analysis*

We want to show the following theorem.

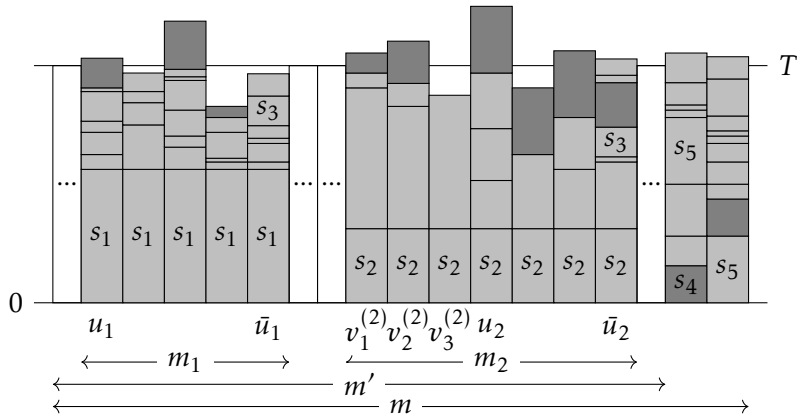


Figure 3.7: The situation after step 4. of Algorithm 3 with $1 \in I_{\exp}$ and $\{2,3,4,5\} \subseteq I_{\text{chp}}$

Theorem 9. *Let I be an instance and let T be a makespan. Let*

$$L_{\text{nonp}} = P(J) + \sum_{i=1}^{c} m_i s_i + \sum_{i:x_i>0} s_i$$

and $m' = \sum_{i=1}^{c} m_i$ where $x_i = P(C_i) - m_i(T - s_i)$. Then the following properties hold.

(i) If $mT < L_{\text{nonp}}$ or $m < m'$, then it is true that $T < \text{OPT}_{\text{nonp}}(I)$.

(ii) Otherwise a feasible schedule with makespan at most $\frac{3}{2}T$ can be computed in time $\mathcal{O}(n)$.

We start with some preliminary work and obtain the following important notes.

Observation 6. *The remaining processing time for class i after step 2. is x_i, i. e. $P(C_i') = x_i$ for all $i \in [c]$ with $x_i \geq 0$. Furthermore, $x_i < 0$ implies that there is a time of $|x_i|$ left to schedule arbitrary jobs.*

Proof. We consider the situation right after step 1.. First we want to know the time F_i that is left to schedule jobs of a class $i \in [c]$ without adding new setups. Each machine of class i got a time of $T - s_i$ to schedule the jobs of $C_i \cap L$. Since there are m_i of such machines we obtain $F_i = m_i(T - s_i) - P(C_i \cap L) \geq 0$. The remaining jobs of class i are $C_i \setminus L$ and that gives us a total residual processing time of $P(C_i \setminus L)$. Let $C_i' \subseteq C_i \setminus L$ be the residual jobs *after* step 2.. Since $C_i = (C_i \setminus L) \dot\cup (C_i \cap L)$, we obtain

$$\begin{aligned} P(C_i') = P(C_i \setminus L) - F_i &= P(C_i \setminus L) + P(C_i \cap L) - m_i(T - s_i) \\ &= P(C_i) - m_i(T - s_i) \ = \ x_i \end{aligned}$$

if $x_i \geq 0$. So if $x_i < 0$, we have $P(C_i \setminus L) < F_i$ and that means there is a time of $F_i - P(C_i \setminus L) = |P(C_i \setminus L) - F_i| = |x_i|$ left to schedule any jobs. □

Observation 7. *A T-feasible schedule needs at least $m_i + 1$ setups to place a class i with $x_i > 0$.*

Proof. By its definition we know that $x_i > 0$ means $P(C_i) > m_i(T - s_i)$. So the obligatory m_i machines (and setups) do not provide enough time to schedule all jobs of class i. Hence at least one additional setup must be placed. □

Proof of Theorem 9. (i). We show that $T \geq \mathrm{OPT}_{\mathrm{nonp}}(I)$ implies that $mT \geq L_{\mathrm{nonp}}$ and $m \geq m'$. So let $T \geq \mathrm{OPT}_{\mathrm{nonp}}(I)$. Then there is a feasible schedule σ with makespan T. Due to Observations 5 and 7 and Lemma 12 we get

$$\begin{aligned} mT \geq \mathrm{L}(\sigma) &\geq P(J) + \sum_{i:x_i \leq 0} m_i s_i + \sum_{i:x_i > 0} (m_i + 1)s_i \\ &= P(J) + \sum_{i=1}^{c} m_i s_i + \sum_{i:x_i > 0} s_i \;= L_{\mathrm{nonp}} \end{aligned}$$

and also Lemma 12 proves that $m \geq \sum_{i=1}^{c} m_i = m'$.

(ii). Let $mT \geq L_{\mathrm{nonp}}$ and $m \geq m'$. Note that step 1. uses $\sum_{i=1}^{c} m_i = m'$ machines and since $m \geq m'$, there are enough machines. Furthermore, one can easily confirm that the wrap templates $\omega^{(i)}$ suffice to schedule the wrap sequences $Q^{(i)}$, i. e. $S(\omega^{(i)}) \geq \mathrm{L}(Q^{(i)})$, but we still need to show that there is enough time to fill up with step 2. and 3.. Instead of analyzing these steps separately we can use the values x_i to find a much more intuitive formalization for both of them. Apparently in general the steps fill up the m' obligatory machines to at least time T. In the worst case they fill them up to *exactly* time T since the residual load of Q, which is to be placed on the residual (and so far unused) $m - m'$ machines, gets maximized then. Due to Observation 6 and Observation 7, this (worst case) residual load R can be written as $R = \sum_{i:x_i<0} x_i + \sum_{i:x_i>0}(s_i + x_i)$ and we show that $R \leq (m - m')T$ as follows.

$$\begin{aligned} R &= \sum_{i=1}^{c} x_i + \sum_{i:x_i>0} s_i \\ &= P(J) - \left(\sum_{i=1}^{c} m_i\right) T + \sum_{i=1}^{c} m_i s_i + \sum_{i:x_i>0} s_i && \text{Def. } x_i \\ &= L_{\mathrm{nonp}} - m'T \\ &\leq (m - m')T && mT \geq L_{\mathrm{nonp}} \end{aligned}$$

So the $m - m'$ residual machines do provide enough time to schedule R. Hence, all load can be placed and it remains to show that step 4. is correct. Apparently step 1. and 2. fill up machines to at most T. Step 3. fills machines to at most $\frac{3}{2}T$. Now consider the situation right after step 3. and remark that the parent jobs j of all preempted jobs of a class i hold $t_j \leq s_i + t_j \leq \frac{1}{2}T$ since $j \in J \setminus L$. The first modification of step 4. is to replace preempted jobs (which are last on a machine with load at most T) with their integral parent jobs while removing all other child pieces. It is easy to see that the makespan can raise up to at most $T + \frac{1}{2}T = \frac{3}{2}T$. Also no jobs are preempted anymore since for each job piece j there was a job piece j' with $\mathrm{parent}(j') = \mathrm{parent}(j)$ such that j' was last on a machine. The second and last modification repairs the lack of setups. Passing the T-crossing items to the next machine u_+ below

the next job will give extra load of at most $\frac{1}{2}T$ to machine u_+ (either passing a setup or a job with an additional setup). For u_+ there are two cases. If u_+ is not the last used machine, then u_+ passes away its last item too such that its load will be at most $\frac{3}{2}T$ after all. If u_+ is the last used machine, it has a load of at most T (otherwise this is a contradiction to $R \leq (m - m')T$) so it will end up with a load of at most $\frac{3}{2}T$. The order of Q guarantees that this movement/addition of setups will remove any lacks of setups such that the resulting schedule is feasible with a makespan of at most $\frac{3}{2}T$.

However, it remains to obtain the running time. The inclined reader will obtain that the primitive way of shifting up items on the considered next machines may require non-linear time, but this can actually be avoided as an implementation detail, with additional running time no more than $\mathcal{O}(n)$. All other steps can be confirmed to run in linear time in a straightforward way. □

3.6 PREEMPTIVE SCHEDULING

One basic tool will be *Batch Wrapping*, i. e. the wrapping of *wrap sequences* into *wrap templates*. See Section 3.2.1 for the short details. For this section let $T_{\min} = \max\{\frac{1}{m}N, \max_{i\in[c]}(s_i + t^{(i)}_{\max})\}$ where $N = \sum_{i=1}^{c} s_i + \sum_{j\in J} t_j$ and $t^{(i)}_{\max} = \max_{j\in C_i} t_j$.

3.6.1 *Nice Instances*

In the following we take a closer look on $I_{\exp}$ and I_{chp} so we split them again. As stated before we divide the expensive classes into three disjoint subsets $I^+_{\exp}$, $I^0_{\exp}$ and $I^-_{\exp}$ such that $i \in I_{\exp}$ holds $i \in I^+_{\exp}$ iff. $T \leq s_i + P(C_i)$, $i \in I^0_{\exp}$ iff. $\frac{3}{4}T < s_i + P(C_i) < T$ and $i \in I^-_{\exp}$ iff. $s_i + P(C_i) \leq \frac{3}{4}T$. Also we divide the cheap classes into I^+_{chp}, I^-_{chp} s.t. $i \in I_{\text{chp}}$ holds $i \in I^+_{\text{chp}}$ iff. $\frac{1}{4}T \leq s_i \leq \frac{1}{2}T$ and $i \in I^-_{\text{chp}}$ iff. $s_i < \frac{1}{4}T$. Now denote the big jobs of a class $i \in I^-_{\text{chp}}$ as $C^*_i = \{j \in C_i \mid s_i + t_j > \frac{1}{2}T\}$ and let $I^*_{\text{chp}} = \{i \in I^-_{\text{chp}} \mid 1 \leq |C^*_i|\} \subseteq I^-_{\text{chp}}$ be the set of classes that contain at least one of these jobs.

Definition 8 (Nice Instances). *For a makespan T we call an instance* nice *if $I^0_{\exp}$ is empty.*

Let $\alpha'_i = \lfloor P(C_i)/(T - s_i) \rfloor \leq \lceil P(C_i)/(T - s_i) \rceil = \alpha_i$ and remark that $\alpha'_i \geq 1$ for all $i \in I^+_{\exp}$. The following theorem will be of great use to find a 1.5-ratio also for general instances.

Theorem 10. *Let I be a nice instance for a makespan T. Moreover, let*

$$L_{\text{nice}} = P(J) + \sum_{i\in I^+_{\exp}} \alpha'_i s_i + \sum_{i\in I^-_{\exp}\cup I_{\text{chp}}} s_i$$

and $m_{\text{nice}} = \lceil \frac{1}{2}|I^-_{\exp}| \rceil + \sum_{i\in I^+_{\exp}} \alpha'_i$ where $\alpha'_i = \left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor$. Then the following properties hold.

(i) If $mT < L_{\text{nice}}$ or $m < m_{\text{nice}}$, it is true that $T < \text{OPT}_{\text{pmtn}}(I)$.

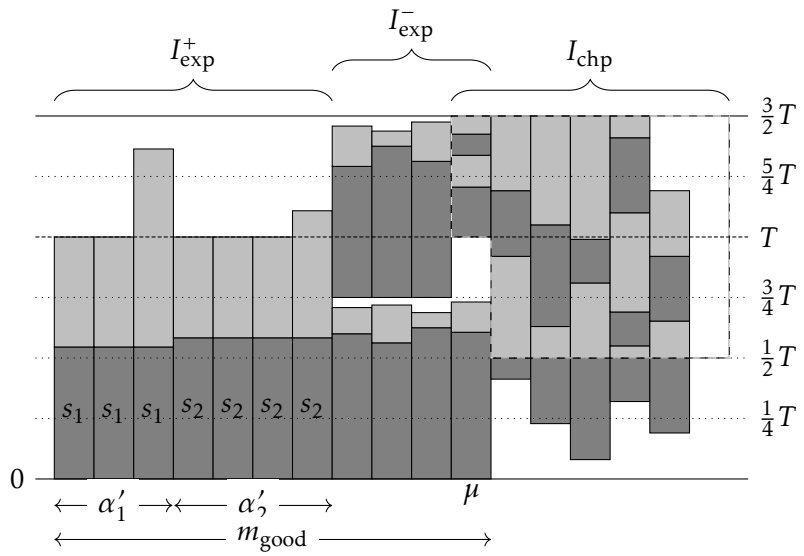


Figure 3.8: An example solution after using Algorithm 4 with $I^+_{\text{exp}} = \{1, 2\}$

(ii) Otherwise a feasible schedule with makespan at most $\frac{3}{2}T$ can be computed in time $\mathcal{O}(n)$.

Algorithm 4 A 1.5-dual Approximation for Nice Instances

1. Schedule $J(I^+_{\text{exp}})$ on $\sum_{i \in I_{\text{exp}}} \alpha'_i$ new machines with α'_i machines for each class i
2. Schedule $J(I^-_{\text{exp}})$ in pairs of two classes on $\lceil \frac{1}{2}|I^-_{\text{exp}}| \rceil$ new machines
3. Wrap $J(I_{\text{chp}})$ onto the residual machines starting on machine μ (last machine of step 2.)

Step 1. First, we look at the classes $i \in I_{\text{exp}}$ with $s_i + P(C_i) \geq T$, i. e. $i \in I^+_{\text{exp}}$. We define a wrap template $\omega^{(i)}$ of length $|\omega^{(i)}| = \lfloor P(C_i)/(T - s_i) \rfloor = \alpha'_i$ for each class $i \in I^+_{\text{exp}}$ as follows. Let $\omega^{(i)}_1 = (u_i, 0, T)$ and $\omega^{(i)}_{1+r} = (u_i + r, s_i, T)$ for all $1 \leq r < \alpha'_i$. The first machines u_i have to be chosen distinct to all machines of the other wrap templates. We construct simple wrap sequences $Q^{(i)} = [s_i, C_i]$ for each class $i \in I^+_{\text{exp}}$ consisting of an initial setup s_i followed by an arbitrary order of all jobs in C_i. For all $i \in I^+_{\text{exp}}$ we use $\text{Wrap}(Q^{(i)}, \omega^{(i)})$ to wrap $Q^{(i)}$ into $\omega^{(i)} = (\omega^{(i)}_1, \ldots, \omega^{(i)}_{\alpha'_i})$. The last machine $\bar{u}_i := u_i + \alpha'_i - 1$ will have a load of at most T but its job load will be less than $\frac{1}{2}T$ since $s_i > \frac{1}{2}T$. We move these jobs to the second last machine and place them on top. So the new load will be greater than T but at most $\frac{3}{2}T$. Finally we remove the setup time s_i on the last machine.

Step 2. Second, we turn to the classes $i \in I_{\text{exp}}$ with $s_i + P(C_i) \leq \frac{3}{4}T$, i. e. $i \in I^-_{\text{exp}}$. We place them paired on one new machine u. So u will have a load $\text{L}(u) = s_{i_1} + P(C_{i_1}) + s_{i_2} + P(C_{i_2})$ for different classes i_1, i_2 that holds

$T = \frac{1}{2}T + \frac{1}{2}T < s_{i_1} + s_{i_2} < \mathrm{L}(u) \leq \frac{3}{4}T + \frac{3}{4}T = \frac{3}{2}T$. Note that the number of such classes can be odd. In this case we schedule one class separate on a new machine μ. Otherwise we choose an unused machine and name it μ. Be aware that this is for the ease of notation; in fact, an unused machine may not exist. So in both cases μ will hold $L(\mu) \leq \frac{3}{4}T$. Apparently this step uses $\lceil \frac{1}{2}|I^-_{\exp}| \rceil$ new machines.

Step 3. The third and last step is to place the jobs of cheap classes. We build a simple wrap template ω as with a case distinction as follows. If $|I^-_{\exp}|$ is odd, we set $\omega_1 = (\mu, T, \frac{3}{2}T)$ and $\omega_{1+r} = (\mu+r, \frac{1}{2}T, \frac{3}{2}T)$ for $1 \leq r \leq m - m_{\text{nice}}$. Otherwise we set $\omega_r = (\mu + r, \frac{1}{2}T, \frac{3}{2}T)$ for all $0 \leq r < m - m_{\text{nice}}$. So in any case we have $|\omega| \leq m - m_{\text{nice}} + 1$. We define Q to be the simple wrap sequence $Q = [s_i, C_i]_{i \in I_{\text{chp}}}$ that contains all jobs of cheap classes. So we wrap Q into $\omega = (\omega_1, \ldots, \omega_{|\omega|})$ using $\textsc{Wrap}(Q, \omega)$.

Figure 3.8 shows an example schedule after the last step. Be aware that setups are dark gray and that jobs of a class are not explicitly drawn.

Proof of Theorem 10. (i). We show that $T \geq \mathrm{OPT}_{\text{pmtn}}(I)$ implies $mT \geq L_{\text{nice}}$ and $m \geq m_{\text{nice}}$. Let $T \geq \mathrm{OPT}_{\text{pmtn}}(I)$. Then there is a feasible schedule σ with makespan T. Let $\mathrm{L}(\sigma) = \sum_{i=1}^{c} (\lambda_i^\sigma s_i + P(C_i))$. Apparently Lemma 5 implies that

$$mT \geq \mathrm{L}(\sigma) \geq P(J) + \sum_{i=1}^{c} \alpha_i s_i \geq P(J) + \sum_{i \in I^+_{\exp}} \alpha'_i s_i + \sum_{i \in I^-_{\exp} \cup I_{\text{chp}}} s_i = L_{\text{nice}}.$$

Due to Lemmas 5 and 6 we know that $m \geq \sum_{i \in I_{\exp}} \lambda_i^\sigma \geq \sum_{i \in I_{\exp}} \alpha_i \geq \sum_{i \in I_{\exp}} \alpha'_i$ and hence

$$m \geq \sum_{i \in I^+_{\exp}} \alpha'_i + \sum_{i \in I^-_{\exp}} \alpha'_i \geq \sum_{i \in I^+_{\exp}} \alpha'_i + \lceil \tfrac{1}{2}|I^-_{\exp}| \rceil = m_{\text{nice}}.$$

(ii). One may easily confirm that the wrap templates $\omega^{(i)}$ suffice to wrap the sequences $Q^{(i)}$ into them, i. e. $S(\omega^{(i)}) \geq \mathrm{L}(Q^{(i)})$ for all $i \in I^+_{\exp}$. Apparently the total number of machines used by step 1. and 2. is $m_{\text{nice}} \leq m$ so there are enough machines for the first two steps. It remains to show that wrap template ω is sufficient to wrap Q into it, i. e. $S(\omega) \geq \mathrm{L}(Q)$. We find that

$$\alpha'_i s_i + P(C_i) \geq \alpha'_i s_i + \left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor (T - s_i) = \alpha'_i s_i + \alpha'_i (T - s_i) = \alpha'_i T \quad (3.2)$$

for all $i \in I^+_{\text{exp}}$. Considering that $|I^-_{\text{exp}}|$ is odd we show the inequality.

$$\begin{aligned}
S(\omega) &= \tfrac{1}{2}T + (m - m_{\text{nice}})T \\
&\geq L_{\text{nice}} + \tfrac{1}{2}T - m_{\text{nice}}T && mT \geq L_{\text{nice}} \\
&= L_{\text{nice}} + \tfrac{1}{2}T - (\lceil \tfrac{1}{2}|I^-_{\text{exp}}| \rceil + \sum_{i \in I^+_{\text{exp}}} \alpha'_i)T \\
&\geq \sum_{i \in I_{\text{chp}}} (s_i + P(C_i)) + \sum_{i \in I^-_{\text{exp}}} (s_i + P(C_i)) + \tfrac{1}{2}T - \lceil \tfrac{1}{2}|I^-_{\text{exp}}| \rceil T && (3.2) \\
&\geq \mathrm{L}(Q) + \sum_{i \in I^-_{\text{exp}}} \tfrac{1}{2}T + \tfrac{1}{2}T - \lceil \tfrac{1}{2}|I^-_{\text{exp}}| \rceil T \;\; = \mathrm{L}(Q)
\end{aligned}$$

This gets even easier if $|I^-_{\text{exp}}|$ is even. □

3.6.2 *General Instances*

Consider a makespan $T \geq T_{\min}$ and remember the partitions $I_{\text{exp}} = I^+_{\text{exp}} \dot\cup I^0_{\text{exp}} \dot\cup I^-_{\text{exp}}$ and $I_{\text{chp}} = I^+_{\text{chp}} \dot\cup I^-_{\text{chp}}$ as well as the machine numbers $\alpha'_i \leq \alpha_i$ as mentioned in Section 3.6.1 on page 57. We state the algorithm and then we go through the details.

Algorithm 5 A 1.5-dual Approximation for Preemptive Scheduling

1. Schedule $J(I^0_{\text{exp}})$ on $l = |I^0_{\text{exp}}|$ machines using one machine per class (the *large machines*)
2. Find the free time F for $J(I^-_{\text{chp}})$ on the residual machines in order to apply Algorithm 4 on a nice instance and split each big job of $J(I^-_{\text{chp}})$ in two pieces (due to Lemma 13 below)
3. Find a feasible placement for $J(I^-_{\text{chp}})$ on the residual $m - l$ empty machines and the free time at the bottom of the large machines of step 1. In more detail:

   If $F < \sum_{i \in I^*_{\text{chp}}} (s_i + P(C_i))$ **then**

   a) Solve an appropriate knapsack instance for the decision, place a nice instance (containing the solution, $J(I^+_{\text{exp}})$, $J(I^-_{\text{exp}})$ and $J(I^+_{\text{chp}})$) with Algorithm 4, and place the unselected items at the bottom of the large machines

   else

   b) Use a greedy approach and the last placement idea of 3.a

Step 1. First we consider all classes $i \in I_{\text{exp}}$ with $\frac{3}{4}T < s_i + P(C_i) < T$, i. e. $i \in I^0_{exp}$. We place every class on its own machine u, i. e. $\mathrm{L}(u) = s_i + P(C_i)$, starting at time $\frac{1}{2}T$. Note that the used machines have less than $\frac{1}{4}T$ free time to schedule any other jobs in a T-feasible schedule. Let $l = |I^0_{\text{exp}}|$ be the number of these machines. We refer to them as the *large machines*. Figure 3.9

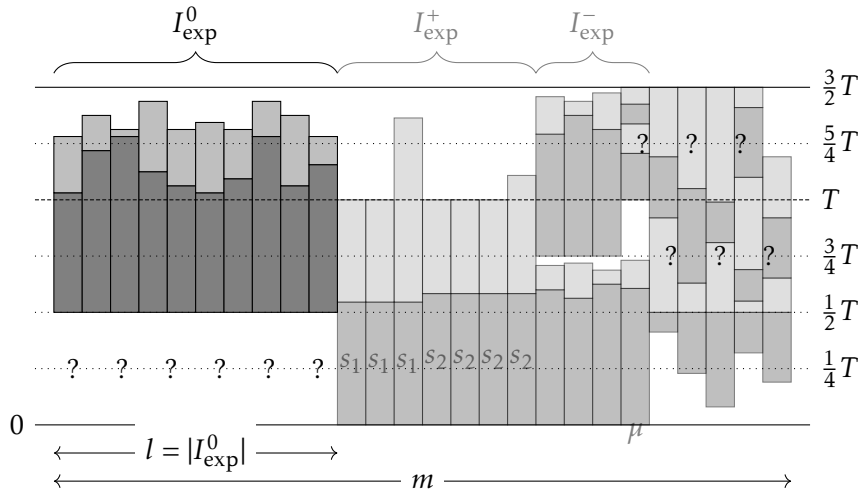


Figure 3.9: An example situation after step 1. with $I^+_{\text{exp}} = \{1, 2\}$

illustrates the situation. The lighter drawn items indicate the future use of Algorithm 4, whereas the question marks indicate the areas where we need to decide the placement of $J(I^-_{\text{chp}})$.

Lemma 13. *In a T-feasible schedule a job $j \in C^*_i$ of a class $i \in I^*_{\text{chp}}$ can not be scheduled on large machines only. Furthermore, j can be processed on large machines with a total processing time of at most $\frac{1}{2}T - s_i$.*

Proof. Remark that the load of the large machines is at least $\frac{3}{4}T$. The placed setup times are greater than $\frac{1}{2}T$ so we can not pause their jobs execution to schedule other jobs because that would need at least one more setup time. So the load of the large machines has to be scheduled consecutively and hence there is at most $\frac{1}{4}T$ time to place other load at the bottom or on top of them. Due to that and since at least one setup s_i is required, it is easy to see that there can be scheduled at most $2(\frac{1}{4}T - s_i) < \frac{1}{2}T - s_i < t_j$ time of job j on large machines (on one at the top and on another one at the bottom) since j is not allowed to be processed on different machines at the same time. □

Step 2. Since the setups of the classes I^+_{chp} are too big to place any of their jobs on a large machine, we definitely will place the jobs $J(I^+_{\text{exp}} \cup I^-_{\text{exp}} \cup I^+_{\text{chp}})$ entirely on the residual $m - l$ machines. We need to obtain the free time F for $J(I^-_{\text{chp}})$ on the residual machines; in fact, we want to place as much load as possible, so, looking at Algorithm 4 we find that the time

$$F = (m-l)T - \sum_{i \in I^+_{\text{exp}}} (\alpha'_i s_i + P(C_i)) - \sum_{i \in I^-_{\text{exp}} \cup I^+_{\text{chp}}} (s_i + P(C_i)) \tag{3.3}$$

can be used to place the jobs of $J(I^-_{\text{chp}})$ in step 3. of Algorithm 4.

Apparently, the free processing time on the large machines sums up to $\tilde{F} := \sum_{u=1}^{l} (T - \mathrm{L}(u))$, so for a suitable value of T the residual available processing time $\tilde{F} + F$ suffices to schedule the residual jobs $J(I^-_{\text{chp}})$. Remember that $C^*_i = \{ j \in C_i \mid s_i + t_j > \frac{1}{2}T \}$ are the big jobs of a class $i \in I^-_{\text{chp}}$, and $I^*_{\text{chp}} \subseteq I^-_{\text{chp}}$

denotes the classes that contain at least one of these jobs. As stated out in Lemma 13, we can not place them on large machines only. So we split them in the following way. For all jobs $j \in C_i^*$ with $i \in I^*_{\text{chp}}$ we create new job pieces $j^{(1)}$ and $j^{(2)}$ with processing times $t_j^{(1)}$ and $t_j^{(2)}$, satisfying $t_j^{(1)} = \frac{1}{2}T - s_i$ as well as $t_j^{(2)} = s_i + t_j - \frac{1}{2}T$ and hence, $t_j^{(1)} + t_j^{(2)} = t_j$. Note that $s_i + t_j^{(1)} \leq \frac{1}{2}T$ and $t_j^{(2)} \leq T - \frac{1}{2}T = \frac{1}{2}T$. Due to Lemma 13 we have to schedule a processing time of at least $t_j^{(2)}$ of job $j \in C_i^*$ *alongside* the large machines. So we also have an obligatoy setup time s_i alongside the large machines. The now following case distinction of step 3. is a bit more complicated.

Case 3.a: $F < \sum_{i \in I^*_{\text{chp}}} (s_i + P(C_i))$. Now we have to use large machines to schedule all jobs of $J(I^*_{\text{chp}})$. The task is to optimize the use of setup times. We do this by minimizing the total load of necessary new setup times to be placed on the large machines $1, \dots, l$. Each class that can be scheduled entirely alongside the large machines will not cause a setup time on large machines. So the setup optimization can be done by maximizing the total sum of setup times of classes we schedule *entirely* alongside large machines. The obligatory job load *alongside* large machines for a class $i \in I^*_{\text{chp}}$ is

$$L_i^* = \sum_{j \in C_i^*} t_j^{(2)} = \sum_{j \in C_i^*} (s_i + t_j - \tfrac{1}{2}T) = P(C_i^*) - |C_i^*|(\tfrac{1}{2}T - s_i). \tag{3.4}$$

Therefore the total obligatory load alongside large machines of all classes in I^*_{chp} is

$$L^* = \sum_{i \in I^*_{\text{chp}}} (s_i + L_i^*) = \sum_{i \in I^*_{\text{chp}}} (s_i + P(C_i^*) - |C_i^*|(\tfrac{1}{2}T - s_i)). \tag{3.5}$$

Now we can interpret the maximization problem as a knapsack problem by setting $\mathcal{I} = I^*_{\text{chp}}$, capacity $Y = F - L^*$, profit $p_i = s_i$ and weight $w_i = P(C_i) - L_i^*$ for all $i \in I^*_{\text{chp}}$. We compute an optimal solution x_{cks} for the *continuous knapsack problem* with split item $e \in I^*_{\text{chp}}$ that leads to a nearly optimal solution x_{ks} for ILP_{ks} (general knapsack problem) computable in time $\mathcal{O}(|I^*_{\text{chp}}|) \leq \mathcal{O}(c)$. So $0 < (x_{\text{cks}})_e < 1$ may be critical because this means we need to schedule an extra setup time s_e although it might not be necessary in an optimal schedule. We overcome this issue later. Remark that $\sum_{i \in I^*_{\text{chp}}} (x_{\text{ks}})_i w_i = Y - (x_{\text{cks}})_e w_e$ so $(x_{\text{cks}})_e w_e$ is the time to fill with job load of class e. Therefore, we create new job pieces as follows. For all $j \in C_e$ let $j^{[1]}$ and $j^{[2]}$ be jobs with processing times $t_j^{[1]}$ and $t_j^{[2]}$ holding

$$t_j^{[2]} = \begin{cases} (x_{\text{cks}})_e t_j & : \quad j \in C_e \setminus C_e^* \\ (x_{\text{cks}})_e t_j^{(1)} + t_j^{(2)} & : \quad j \in C_e^* \end{cases} \tag{3.6}$$

as well as $t_j^{[1]} = t_j - t_j^{[2]}$. Now we define a new instance $I^{(\text{new})}$ containing all classes $i \in I^+_{\text{exp}} \cup I^-_{\text{exp}} \cup I^+_{\text{chp}}$ with all of their jobs $C_i^{(\text{new})} := C_i$, all selected

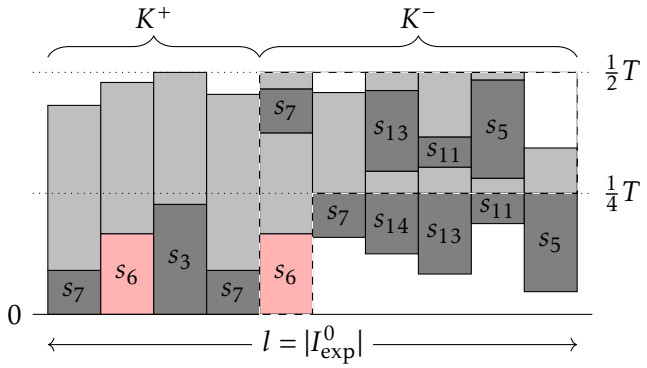


Figure 3.10: An example solution at the bottom of the large machines after using Algorithm 5 with $\{3,5,6,7,11,13,14\} \subseteq I^-_{\text{chp}}$ and $e = 6$

classes $i \in I^*_{\text{chp}}$ holding $(x_{\text{cks}})_i = 1$ with all of their jobs $C_i^{(\text{new})} := C_i$, all unselected classes $i \in I^*_{\text{chp}} \setminus \{e\}$ holding $(x_{\text{cks}})_i = 0$ with just their obligatory load $C_i^{(\text{new})} := \{j^{(2)} \mid j \in C_i^*\}$, and the split item class $e \in I^*_{\text{chp}}$ with just the load $C_e^{(\text{new})} := \{j^{[2]} \mid j \in C_e\}$. Last we set $m^{(\text{new})} = m - l$. Apparently $I^{(\text{new})}$ is a nice instance and we schedule it on the residual $m^{(\text{new})}$ machines using Algorithm 4. Later we will see that this load fills the gap of $(x_{\text{cks}})_e w_e$ to Y since the obligatory job load of $\sum_{j \in C_e^*} t_j^{(2)}$ is enlarged by exactly $(x_{\text{cks}})_e w_e$. So with x_{cks} we found a (sub-)schedule that fills up the free time Y alongside the large machines in an optimal way; in fact, we maximized the setup times of the selected classes such that the sum of the setup times of unselected classes got minimized. Hence, the residual load can be scheduled feasibly in the free time $\tilde{F}$ on the large machines, if T is suitable. Let K be the set of the residual jobs and job pieces, i. e.

$$K = \{j^{[1]} \mid j \in C_e\} \cup \bigcup_{\substack{i \in I^*_{\text{chp}} \setminus \{e\} \\ (x_{\text{cks}})_i = 0}} \left(\{j^{(1)} \mid j \in C_i^*\} \cup (C_i \setminus C_i^*) \right) \cup J(I^-_{\text{chp}} \setminus I^*_{\text{chp}}). \quad (3.7)$$

In Observation 8 we will see that all jobs (or job pieces) $\iota \in K$ of a class i hold $s_i + t_\iota \leq \frac{1}{2}T$. In the following we schedule the jobs of K at the bottom of the large machines.

Observation 8. *All jobs (or job pieces) $\iota \in K$ of class i hold $s_i + t_\iota \leq \frac{1}{2}T$.*

Proof. First we show that this is true for all $\iota \in \{j^{[1]} \mid j \in C_e\}$. For $j \in C_e \setminus C_e^*$ we have $s_e + t_j \leq \frac{1}{2}T$ and hence

$$s_e + t_j^{[1]} = s_e + t_j - (x_{\text{cks}})_e t_j \leq \tfrac{1}{2}T - (x_{\text{cks}})_e t_j \leq \tfrac{1}{2}T.$$

For $j \in C_e^*$ we use $t_j^{(2)} = s_e + t_j - \frac{1}{2}T$ to see that

$$s_e + t_j^{[1]} = s_e + t_j - (x_{\text{cks}})_e t_j^{(1)} - (s_e + t_j - \tfrac{1}{2}T) = \tfrac{1}{2}T - (x_{\text{cks}})_e t_j^{(1)} \leq \tfrac{1}{2}T.$$

Let $i \in I^*_{\text{chp}}$ be a class with $(x_{\text{cks}})_i = 0$. We have $s_i + t_j^{(1)} = s_i + \frac{1}{2}T - s_i = \frac{1}{2}T$ for all $j \in C_i^*$ and per definition of C_i^* the bound holds for all jobs in $C_i \setminus C_i^*$. Also as a direct result of the definition of I^*_{chp} we get the bound for all jobs of $J(I^-_{\text{chp}} \setminus I^*_{\text{chp}})$. □

We split $K = K^+ \dot\cup K^-$ into big jobs $K^+ = \{\iota \in K \mid t_\iota > \frac{1}{4}T\}$ and small jobs $K^- = \{\iota \in K \mid t_\iota \leq \frac{1}{4}T\}$. Due to Lemma 15 it suffices to fill large machines with an obligatory load of at least T. Since the large machines already have a load of at least $\frac{3}{4}T$, it is enough to add an obligatory load of at least $\frac{1}{4}T$. We start with the jobs of K^+. On the one hand all $\iota \in K^+$ of a class i hold $t_\iota > \frac{1}{4}T$ and on the other hand they can be placed entirely at the bottom of a large machine since $s_i + t_\iota \leq \frac{1}{2}T$. So this is what we do. We place the jobs of K^+ on the first $l' \leq l$ large machines $1, \ldots, l'$ with an initial associated setup time at time 0 directly followed by the job (or job piece). The very last step is to schedule the jobs of K^-. We remember that we need to schedule one setup time s_e extra iff $(x_{\text{cks}})_e > 0$. To avoid a case distinction we define a wrap template which is slightly larger than required for the obligatory load. Since all jobs (or job pieces) $\iota \in K^-$ need at most $\frac{1}{4}T$ time, they can be wrapped without parallelization using a wrap template ω with $|\omega| = l - l'$ defined by $\omega_1 = (l' + 1, 0, \frac{1}{2}T)$ and $\omega_{1+r} = (l' + 1 + r, \frac{1}{4}T, \frac{1}{2}T)$ for all $1 \leq r < l - l'$. To construct a wrap sequence Q, we order the jobs and/or job pieces in K^- by class, beginning with class e, and insert a suitable setup before the jobs of each class. Remark that Q starts with s_e followed by an arbitrary order of $\{j^{[1]} \mid j \in C_e\} \cap K^-$. Finally we wrap Q into ω using $\textsc{Wrap}(Q, \omega)$.

See Figure 3.10 for an example solution at the bottom of the large machines. The split item class setup e is colored red. Note that we even got two setups $s_6 = s_e$ since there was a big job in $K^+ \cap \{j^{[1]} \mid j \in C_e\}$. Also notice that setup s_{14} was a critical item on machine 6 and therefore it was moved below the next gap.

Case 3.b: $F \geq \sum_{i \in I^*_{\text{chp}}}(s_i + P(C_i))$. Then there is enough time to schedule the jobs $J(I^*_{\text{chp}})$ entirely alongside the large machines. Taking the previous case as a more complex model for this one, split $J(I^-_{\text{chp}} \setminus I^*_{\text{chp}})$ into two well-defined wrap sequences Q_1, Q_2 such that $\text{L}(Q_1) = F - \sum_{i \in I^*_{\text{chp}}}(s_i + P(C_i))$ and there is at most one class $e \in I^-_{\text{chp}} \setminus I^*_{\text{chp}}$ with jobs (or job pieces) in *both* sequences. Such a splitting can be obtained by a simple greedy approach. Then $J(I^+_{\text{exp}} \cup I^-_{\text{exp}} \cup I^*_{\text{chp}})$ and the job pieces of Q_1 lead to a nice instance for the residual $m - l$ machines while the job pieces of Q_2 can be named K, be split into $K = K^+ \dot\cup K^-$ and be handled just like before.

3.6.3 *Analysis*

We study case 3.a ($F < \sum_{i \in I^*_{\text{chp}}}(s_i + P(C_i))$) only since the opposite case is much easier. We want to show the following theorem.

Theorem 11. *Let I be an instance and let T be a makespan. Let $\alpha'_i = \left\lfloor \frac{P(C_i)}{T-s_i} \right\rfloor$ for all $i \in I^+_{\text{exp}}$ and x_{cks} be the optimal solution to the knapsack problem of step 3. and let*

$$L_{\text{pmtn}} = P(J) + \sum_{i \in I^+_{\text{exp}}} \alpha'_i s_i + \sum_{i \in [c] \setminus I^+_{\text{exp}}} s_i + \sum_{\substack{i \in I^*_{\text{chp}} \\ (x_{\text{cks}})_i = 0}} s_i$$

and $m' = |I^0_{\text{exp}}| + \sum_{i \in I^+_{\text{exp}}} \alpha'_i + \lceil \frac{1}{2} |I^-_{\text{exp}}| \rceil$. Then the following properties hold.

(i) If $mT < L_{\text{pmtn}}$ or $m < m'$, then it is true that $T < \text{OPT}_{\text{pmtn}}(I)$.

(ii) Otherwise a feasible schedule with makespan at most $\frac{3}{2}T$ can be computed in time $\mathcal{O}(n)$.

Proof. (i). We show that $T \geq \text{OPT}_{\text{pmtn}}$ implies $mT \geq L_{\text{pmtn}}$ and $m \geq m'$. Let $T \geq \text{OPT}_{\text{pmtn}}$. Then there is a feasible schedule σ with makespan T. Let $\text{L}(\sigma) = \sum_{i=1}^{c} (\lambda^\sigma_i s_i + P(C_i))$. Since $F < \sum_{i \in I^*_{\text{chp}}} (s_i + P(C^*_i))$, we know that we will need an extra setup s_i for all unselected classes $i \in I^*_{\text{chp}}$ holding $(x_{\text{cks}})_i = 0$, due to Lemma 13. Together with Lemma 5 we get that

$$\begin{aligned} mT \geq \text{L}(\sigma) &\geq P(J) + \sum_{i=1}^{c} \alpha_i s_i \\ &\geq P(J) + \sum_{i \in I^+_{\text{exp}}} \alpha'_i s_i + \sum_{i \in [c] \setminus I^+_{\text{exp}}} s_i + \sum_{\substack{i \in I^*_{\text{chp}} \\ (x_{\text{cks}})_i = 0}} s_i = L_{\text{pmtn}}. \end{aligned}$$

Due to Lemmas 5 and 6 we know that $m \geq \sum_{i \in I_{\text{exp}}} \lambda^\sigma_i \geq \sum_{i \in I_{\text{exp}}} \alpha_i$ and hence

$$m \geq \sum_{i \in I^0_{\text{exp}}} \alpha_i + \sum_{i \in I^+_{\text{exp}}} \alpha_i + \sum_{i \in I^-_{\text{exp}}} \alpha_i \geq |I^0_{\text{exp}}| + \sum_{i \in I^+_{\text{exp}}} \alpha'_i + \lceil \tfrac{1}{2} |I^-_{\text{exp}}| \rceil = m'.$$

(ii). Let $mT \geq L_{\text{pmtn}}$ and $m \geq m'$. Apparently there are enough machines $m \geq m' \geq |I^0_{\text{exp}}| = l$ for step 1.. It is important to see that this simple placement of one machine per class is legitimate only due to Lemma 15. Now we study step 2.. As mentioned in the description of the algorithm, $I^{(\text{new})}$ is a nice instance but we want to see that it is placed feasibly on the last $m-l$ machines. So we look on the split item class $e \in I^*_{\text{chp}}$ again. We find that

$$\begin{aligned} \sum_{j \in C_e} t^{[2]}_j &= \sum_{j \in C_e \setminus C^*_e} (x_{\text{cks}})_e t_j + \sum_{j \in C^*_e} \left((x_{\text{cks}})_e t^{(1)}_j + t^{(2)}_j \right) \\ &= \sum_{j \in C^*_e} t^{(2)}_j + (x_{\text{cks}})_e \left(\sum_{j \in C_e \setminus C^*_e} t_j + \sum_{j \in C^*_e} t^{(1)}_j \right) \\ &= L^*_e + (x_{\text{cks}})_e \left(\sum_{j \in C_e} t_j - \sum_{j \in C^*_e} t^{(2)}_j \right) && (3.4),\ t^{(1)}_j = t_j - t^{(2)}_j \\ &= L^*_e + (x_{\text{cks}})_e w_e && w_e = P(C_e) - \textstyle\sum_{j \in C^*_e} t^{(2)}_j \end{aligned}$$

and this means that the jobs $\{j^{[2]} \mid j \in C_e\}$ do expand the obligatory load L^*_e alongside the large machines of e by exactly $(x_{\text{cks}})_e w_e$, as mentioned before. Turning back to instance I^{new}, we name the cheap load $L^{(\text{new})}_{\text{chp}}$ and find that

$$L^{(\text{new})}_{\text{nice}} = \sum_{i \in I^+_{\text{exp}}} (\alpha'_i s_i + P(C_i)) + \sum_{i \in I^-_{\text{exp}} \cup I^+_{\text{chp}}} (s_i + P(C_i)) + L^{(\text{new})}_{\text{chp}}. \tag{3.8}$$

We use the continuous knapsack characteristic $\sum_{i \in I^*_{\text{chp}}} (x_{\text{cks}})_i w_i + (x_{\text{cks}})_e w_e = Y = F - L^*$ as well as $w_i = P(C_i) - L^*_i$ and hence $P(C_i) = L^*_i + w_i$ to show the following equality.

$$\begin{aligned}
L^{(\text{new})}_{\text{chp}} &= \sum_{\substack{i \in I^*_{\text{chp}} \\ (x_{\text{cks}})_i = 1}} (s_i + P(C_i)) + \sum_{\substack{i \in I^*_{\text{chp}} \setminus \{e\} \\ (x_{\text{cks}})_i = 0}} (s_i + P(\{j^{(2)} \mid j \in C^*_i\})) \\
&\qquad\qquad + s_e + P(\{j^{[2]} \mid j \in C_e\}) \\
&= \sum_{\substack{i \in I^*_{\text{chp}} \\ (x_{\text{cks}})_i = 1}} (s_i + L^*_i + w_i) + \sum_{\substack{i \in I^*_{\text{chp}} \setminus \{e\} \\ (x_{\text{cks}})_i = 0}} (s_i + L^*_i) + s_e + L^*_e + (x_{\text{cks}})_e w_e \\
&= \sum_{i \in I^*_{\text{chp}}} (s_i + L^*_i) + \sum_{\substack{i \in I^*_{\text{chp}} \\ (x_{\text{cks}})_i = 1}} w_i + (x_{\text{cks}})_e w_e \\
&= L^* + \sum_{i \in I^*_{\text{chp}}} (x_{\text{cks}})_i w_i + (x_{\text{cks}})_e w_e \;=\; L^* + F - L^* \;=\; F
\end{aligned}$$

So with Equations (3.5) and (3.8) it follows directly that $m^{(\text{new})} T = (m-l)T = L^{(\text{new})}_{\text{nice}}$. Also we have

$$m^{(\text{new})} = m - l \geq m' - l = l + \sum_{i \in I^+_{\text{exp}}} \alpha'_i + \lceil \tfrac{1}{2} |I^-_{\text{exp}}| \rceil - l = m_{\text{nice}}$$

so Theorem 10(ii) is satisfied and hence I^{new} is scheduled feasibly on the last $m - l$ machines with a makespan of at most $\frac{3}{2}T$. It remains to show that K can be placed at the bottom of the large machines. As already stated in the description, this is ensured by the optimality of the continuous knapsack solution. One may formally confirm that

$$l \cdot \tfrac{1}{4} T \geq \tilde{F} \geq P(K) + \sum_{\substack{i \in I^*_{\text{chp}} \\ (x_{\text{cks}})_i = 0}} s_i + \sum_{i \in I^-_{\text{chp}} \setminus I^*_{\text{chp}}} s_i.$$

Apparently each job in K^+ has a load of at least $\frac{1}{4}T$ and is placed on exactly one large machine u, which holds $T - \mathrm{L}(u) < \frac{1}{4}T$. According to this, the wrap template ω suffices to wrap the residual jobs K^-. Even big jobs of the split item class (or its setups) are no problem, since big jobs fill up large machines more than possible (in a T-feasible schedule). So the only setup to worry about is the setup s_e wrapped into ω. One can see that it is not part of the

above inequality, since there is no time reserved for it. Fortunately, the time period $S(\omega)$ provided by ω is large enough, though. This is true, since

$$S(\omega) = \tfrac{1}{2}T + (l - l' - 1)\tfrac{1}{4}T = (l - l' + 1)\tfrac{1}{4}T \\ \geq (l - |K^+|)\tfrac{1}{4}T + s_e = |K^-|\tfrac{1}{4}T + s_e.$$

Remark that jobs do never run in parallel, according to sufficient gap heights. However, it remains to obtain the running time. The total sum of the lengths of all used wrap templates *and* wrap sequences is in $\mathcal{O}(n)$. So they are wrapped in a total time of $\mathcal{O}(n)$. Also we need at time of $\mathcal{O}(n)$ to compute the knapsack instance. To solve it, we have linear time again. Overall the running time is $\mathcal{O}(n)$. □

3.6.4 *Class Jumping*

Here we use our idea of *Class Jumping* (cf. Section 3.3.4) to show the following theorem.

Theorem 12. *There is a* 1.5*-approximation for the preemptive case running in time* $\mathcal{O}(n \log n)$.

The idea for the splittable case can be applied to the preemptive one with a small modification as follows. We need to replace step 1. of Algorithm 4 on page 58 such that the jumps of I^+_{exp} depend less on the setup time s_i. As in the algorithm for the splittable case, we define a gap of size $\frac{1}{2}T$ above each setup. If a last machine got a total load of at least T the machines are filled well. If a last machine got load less than T its job load will be at most $T - s_i$. It turns out that the machines u before hold $\frac{3}{2}T - \mathrm{L}(u) = \frac{3}{2}T - (s_i + \frac{1}{2}T) = T - s_i$. So we simply move the job load of the last machine to the top of the second last machine (and remove the setup on the last machine). See Figure 3.11 for an example. To define the associated machine number γ_i for all classes $i \in I^+_{\text{exp}}$ we set

$$\beta'_i = \left\lfloor \frac{2P(C_i)}{T} \right\rfloor \quad \text{and} \quad \gamma_i = \begin{cases} \max\{\beta'_i, 1\} & P(C_i) - \beta'_i \cdot \frac{1}{2}T \leq T - s_i \\ \beta_i & \text{otherwise} \end{cases}.$$

Remark that $\gamma_i \leq \beta_i$. One can see that class i jumps right after the last machine got a job load of exactly $T - s_i$. In more detail we obtain that a makespan T is a jump of class i if $P_i = \gamma_i(T) \cdot \frac{1}{2}T + (T - s_i)$ and this may be rearranged to $T = 2(s_i + P_i)/(\gamma_i(T) + 2)$.

ANALYSIS OF ALGORITHM 6 Again we focus on the crucial claim, that X contains no more than c jumps in total.

Lemma 14. *If T' is a jump of f, i. e. $T' = 2(s_f + P_f)/(\gamma_f(T') + 2)$, and T'' is a jump of a different class i, i. e. $T'' = 2(s_i + P_i)/(\gamma_i(T'') + 2)$, such that $T'' \leq T'$, then the next jump of class i is smaller than the next jump of f, which may be written as*

$$\frac{2(s_i + P_i)}{\gamma_i(T'') + 3} \leq \frac{2(s_f + P_f)}{\gamma_f(T') + 3}.$$

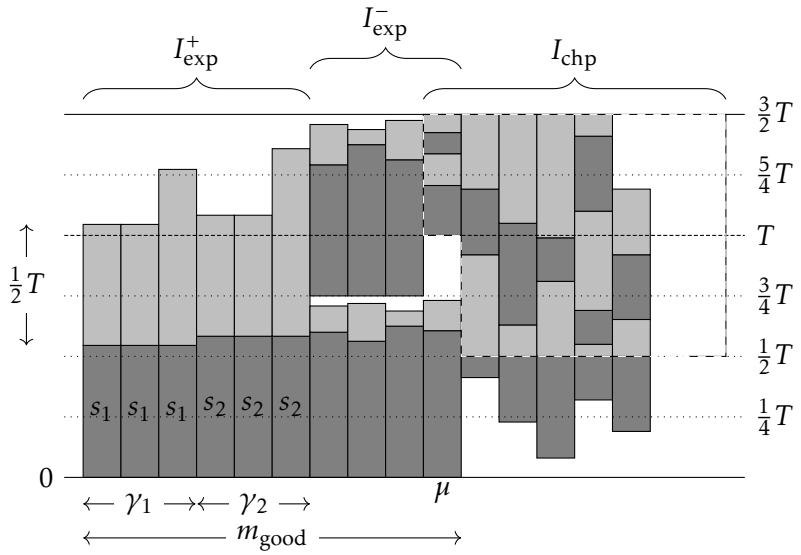


Figure 3.11: An example solution after using the modification of Algorithm 4 with $I^+_{\text{exp}} = \{1, 2\}$

Algorithm 6 Class Jumping for Preemptive Scheduling

1. Compute $P_i = P(C_i)$ for all $i \in [c]$ in time $\mathcal{O}(n)$
2. Use consecutive binary search routines to find a right interval $V = (A_1, T_1]$ in time $\mathcal{O}(n \log c)$ such that each $T \in (A_1, T_1]$ causes the same sets I^+_{exp}, I^0_{exp}, I^-_{exp} and $\{i \in I^*_{\text{chp}} \mid (x_{\text{cks}})_i = 0\}$
3. Find a *fastest jumping class* $f \in I^+_{\text{exp}}$, i. e. $s_f + P_f \geq s_i + P_i$ for all $i \in I^+_{\text{exp}}$, in time $\mathcal{O}(c)$
4. Guess the right interval $W = \left(\frac{2(s_f+P_f)}{\gamma_f(T_1)+k+3}, \frac{2(s_f+P_f)}{\gamma_f(T_1)+k+2}\right]$ for some integer $k < m$ such that $X := V \cap W \neq \emptyset$, in time $\mathcal{O}(c \log m)$ and set $T_2 = \frac{2(s_f+P_f)}{\gamma_f(T_1)+k+2}$
5. Find and sort the $\mathcal{O}(c)$ jumps in X in time $\mathcal{O}(c \log c)$
6. Guess the right interval $Y = (T_{\text{fail}}, T_{\text{ok}}] \subseteq W$ for two jumps $T_{\text{fail}}, T_{\text{ok}}$ of two classes $i_a, i_b \in I^+_{\text{exp}}$ which jump in X such that no *other* class jumps in Y, in time $\mathcal{O}(c \log c)$
7. Choose a suitable makespan in this interval in constant time

Proof. By simply rearranging the equations for T' and T'' we find that

$$\gamma_f(T') = \frac{2(s_f + P_f)}{T'} - 2 \quad \text{and} \quad \gamma_i(T'') = \frac{2(s_i + P_i)}{T''} - 2$$

and therefore we get

$$\begin{aligned}
\frac{2(s_i + P_i)}{\gamma_i(T'') + 3} = \frac{2(s_i + P_i)}{\frac{2(s_i+P_i)}{T''} - 2 + 3} &= \frac{1}{\frac{1}{T''} + \frac{1}{2(s_i+P_i)}} \\
&\leq \frac{1}{\frac{1}{T'} + \frac{1}{2(s_i+P_i)}} && T'' \leq T' \\
&\leq \frac{1}{\frac{1}{T'} + \frac{1}{2(s_f+P_f)}} \\
&= \frac{2(s_f + P_f)}{\frac{2(s_f+P_f)}{T'} - 2 + 3} = \frac{2(s_f + P_f)}{\gamma_f(T') + 3}.
\end{aligned}$$

□

Proof of Theorem 12. Due to Lemma 14 any class that jumps in X jumps *outside* of X the next time. So for every class $i \in I^+_{\text{exp}}$ there is at most one jump in X and hence X contains at most $|I^+_{\text{exp}}| \leq c$ jumps. Apparently the sum of the running times of each step is $\mathcal{O}(n \log(c + m))$ and the returned value $T \in \{T_{\text{ok}}, T_{\text{new}}\}$ holds $T \leq \text{OPT}_{\text{pmtn}}$ while being accepted by Theorem 11 (ii) such that Algorithm 5 computes a feasible schedule with ratio 3/2 in time $\mathcal{O}(n)$. So we get a total running time of $\mathcal{O}(n \log(c + m)) \leq \mathcal{O}(n \log n)$ since $c \leq n$ and $m < n$. □

3.6.5 *The Soundness of Large Machines*

The proof of Theorem 11 implicitly uses the fact that the use of our large machines is reasonable. To be convinced we prove the following Lemma.

Lemma 15. *For every feasible schedule with makespan T there is a feasible schedule σ with a makespan of at most $\frac{3}{2}T$ such that each class $i \in I^0_{\text{exp}}$ is placed on exactly one machine u_i which holds $s_i + P(C_i) \leq \text{L}_\sigma(u_i) \leq T$.*

We start with a more simple property.

Lemma 16. *Let σ be a feasible schedule with makespan T. Then there is a feasible schedule σ' with a makespan of at most $\frac{3}{2}T$ that holds the following properties.*

1. *If a class $i \in I^0_{\text{exp}}$ is scheduled on exactly one machine u_i in σ (i. e. $\lambda^\sigma_i = 1$) then it is scheduled on exactly one machine u'_i in σ' such that setup s_i starts processing at time $\frac{1}{2}T$ and there is no more load above C_i while $\text{L}'(u'_i) = \text{L}(u_i) \leq T$.*
2. *On all other machines of σ' no job (piece) starts processing before time $\frac{1}{2}T$.*

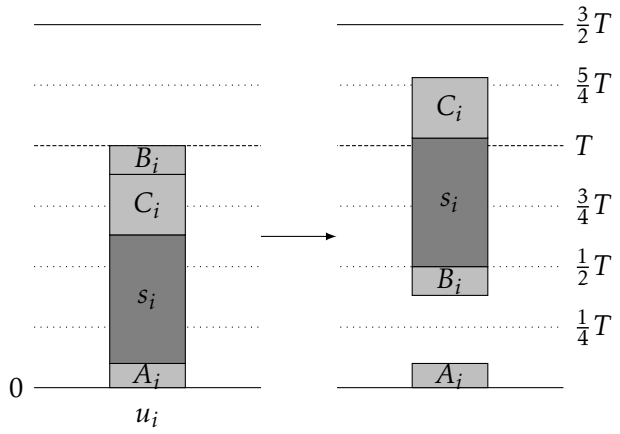


Figure 3.12: Modification of a large machine u_i

Proof. Let σ be a feasible schedule with makespan T. Let $\mathrm{L}(\sigma) = \sum_{i=1}^{c}(\lambda_i^\sigma s_i + P(C_i))$. We do a simple machine modification. Consider a machine u_i in σ that schedules a class $i \in I_{\exp}^0$ holding $\lambda_i^\sigma = 1$. We reorder machine u_i as follows.

We refer to the load below setup time s_i as set A_i and to the load above the last job of C_i as set B_i. So A_i and B_i hold setup times, jobs and job pieces of classes $i' \neq i$. Now move up the setup time s_i as well as all jobs of C_i such that s_i starts at time $\frac{1}{2}T$ and the jobs of C_i are scheduled consecutively right behind it. Also we move down each item of B_i by exactly $\frac{1}{2}T$ such that A_i and B_i are scheduled until time $\frac{1}{2}T$. Since $s_i + P(C_i) > \frac{3}{4}T$, we get $\mathrm{L}(A_i) + \mathrm{L}(B_i) < \frac{1}{4}T$ and thus it follows $\mathrm{L}(A_i) < \frac{1}{4}T$ as well as $\mathrm{L}(B_i) < \frac{1}{4}T$. Hence the items of A_i and B_i do not intersect in time after this modification. Also notice that for different classes $i \neq i'$ there is no forbidden parallelization for preempted jobs of $A_i \cup A_{i'} \cup B_i \cup B_{i'}$ because relatively to each other they are scheduled in time just as before. On all other machines of σ we move up every item by exactly $\frac{1}{2}T$ such that nothing is scheduled before line $\frac{1}{2}T$. Apparently we get a feasible schedule with a makespan of at most $\frac{3}{2}T$. □

Proof of Lemma 15. Let σ be a feasible schedule with makespan T and let $\mathrm{L}(\sigma) = \sum_{i=1}^{c}(\lambda_i^\sigma s_i + P(C_i))$. We add some notation. We set $I_{\exp}^{0,1}(\sigma) \subseteq I_{\exp}^0$ as the set of classes $i \in I_{\exp}^0$ with $\lambda_i^\sigma = 1$ whereas $I_{\exp}^{0,2}(\sigma) = I_{\exp}^0 \setminus I_{\exp}^{0,1}(\sigma)$ denotes the set of classes $i \in I_{\exp}^0$ holding $\lambda_i^\sigma \geq 2$ such that $I_{\exp}^0 = I_{\exp}^{0,1}(\sigma) \dot\cup I_{\exp}^{0,2}(\sigma)$. So $I_{\exp}^{0,1}(\sigma)$ is the set of classes already placed like intended. Nevertheless we need to modify their placement according to the feasibility of other classes. So for all $i \in I_{\exp}^{0,1}(\sigma)$ there is exactly one machine u_i that schedules all jobs of C_i in schedule σ. We modify them using Lemma 16. Apparently these machines may already schedule jobs or job pieces of I_{chp}. To identify the residual load of I_{chp} we do the following. Using the notation of Lemma 16, let $t_j^{(1)}$ be the sum of the processing times of all job pieces in $\bigcup_{i \in I_{\exp}^{0,1}(\sigma)} X_i$ of a job $j \in J(I_{\mathrm{chp}})$ where $X_i = A_i \cup B_i$ for all $i \in I_{\exp}^{0,1}(\sigma)$. Remember that X_i does not contain jobs or job pieces of class i. We create a new job piece $j^{(2)}$ for all jobs $j \in J(I_{\mathrm{chp}})$ with processing time $t_j^{(2)} = t_j - t_j^{(1)}$. So the residual

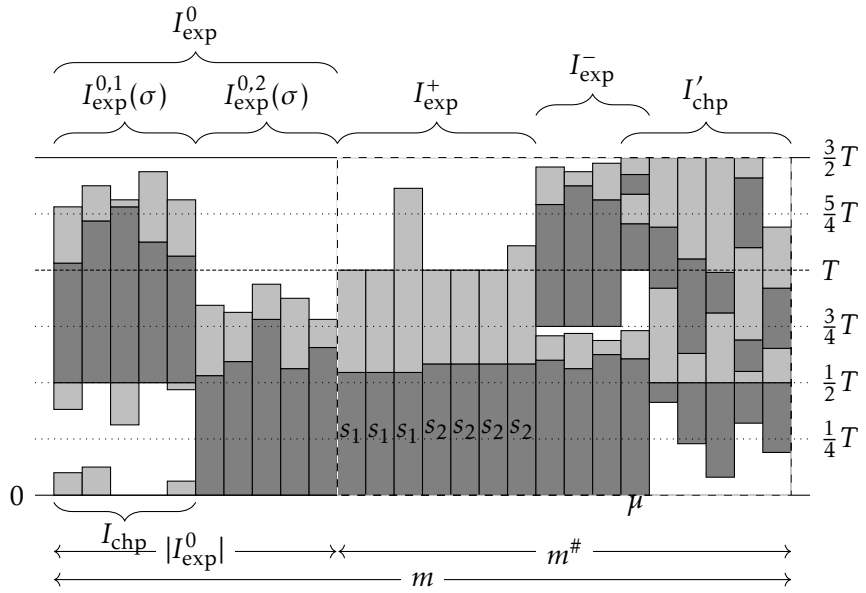


Figure 3.13: An example solution after using Lemma 15 with $I^+_{\text{exp}} = \{1, 2\}$

jobs and job pieces of I_{chp} are $C'_i := \{ j^{(2)} \mid j \in C_i, t^{(2)}_j > 0 \}$ for all $i \in I_{\text{chp}}$. Let $I'_{\text{chp}} = \{ i \in I_{\text{chp}} \mid 1 \leq |C'_i| \}$ and since σ is feasible, we obtain

$$\begin{aligned}(m - |I^{0,1}_{\text{exp}}(\sigma)|)T &\geq \mathrm{L}(\sigma) - \sum_{i I^{0,1}_{\text{exp}}(\sigma)} \mathrm{L}(u_i) \\ &= \sum_{i \in (I_{\text{exp}} \setminus I^{0,1}_{\text{exp}}(\sigma))} (\lambda_i s_i + P(C_i)) + \sum_{i \in I'_{\text{chp}}} (s_i + P(C'_i)).\end{aligned} \tag{3.9}$$

Each one of the $\lambda_i \geq 2$ machines used to schedule the jobs of a class $i \in I^{0,2}_{\text{exp}}(\sigma)$ in σ has a total load of different classes $i' \neq i$ of at most $\frac{1}{2}T$ since $s_i > \frac{1}{2}T$. We aim to schedule them on a single machine such that $\lambda'_i = 1$ if λ'_r is the number of setup times used to schedule class r in schedule σ'. In fact, we can do this without scheduling load of different classes on the selected (single) machines. We extend the schedule as follows. Each class $i \in I^{0,2}_{\text{exp}}$ is placed on a single machine with an initial setup time s_i followed by C_i with no load of other classes underneath or above. At first glance this seems rather wasteful because in schedule σ there may be other load on machines scheduling $I^{0,2}_{\text{chp}}(\sigma)$ in general. With a closer look we can convince us that its reasonable though. The idea is the following. A class $i \in I^{0,2}_{\text{exp}}(\sigma)$ was placed in schedule σ with a load of $L_i = \lambda^\sigma_i s_i + P(C_i) \geq 2s_i + P(C_i)$ on $\lambda^\sigma_i \geq 2$ machines whereas we sum up to a load of $L'_i = s_i + P(C_i)$ on only one machine in schedule σ' now. Hence, we get at least one $s_i > \frac{1}{2}T$ of processing time on a *different* and so far unused machine since there cannot be two setup times of expensive classes on one machine. So we waste a time of $T - (s_i + P(C_i)) < \frac{1}{4}T$ to schedule class i while gaining at least $s_i > \frac{1}{2}T$ of processing time.

To look at this issue in more detail we call R the processing time of the residual load in σ and we find

$$R = \sum_{i \in I^+_{\exp} \cup I^-_{\exp}} (\lambda^\sigma_i s_i + P(C_i)) + \sum_{i \in I'_{\text{chp}}} (s_i + P(C'_i)).$$

Applying $\lambda^\sigma_i \geq 2$ for all $i \in I^{0,2}_{\exp}(\sigma)$ we can use

$$\sum_{i \in I^{0,2}_{\exp}(\sigma)} (\lambda^\sigma_i s_i + P(C_i)) \geq \sum_{i \in I^{0,2}_{\exp}(\sigma)} (2s_i + P(C_i)) \geq \sum_{i \in I^{0,2}_{\exp}(\sigma)} (T + P(C_i)) \geq |I^{0,2}_{\exp}(\sigma)|T$$

and $I_{\exp} \setminus I^{0,1}_{\exp}(\sigma) = I^{0,2}_{\exp}(\sigma) \dot\cup I^+_{\exp} \dot\cup I^-_{\exp}$ to see that

$$\begin{aligned}
&(m - |I^{0,1}_{\exp}(\sigma)|)T \\
&\geq \sum_{i \in (I_{\exp} \setminus I^{0,1}_{\exp}(\sigma))} (\lambda^\sigma_i s_i + P(C_i)) + \sum_{i \in I'_{\text{chp}}} (s_i + P(C'_i)) \qquad (3.9) \\
&= \sum_{i \in I^{0,2}_{\exp}(\sigma)} (\lambda^\sigma_i s_i + P(C_i)) + \sum_{i \in I^+_{\exp} \cup I^-_{\exp}} (\lambda^\sigma_i s_i + P(C_i)) + \sum_{i \in I'_{\text{chp}}} (s_i + P(C'_i)) \\
&\geq |I^{0,2}_{\exp}(\sigma)|T + R
\end{aligned}$$

and thus it follows $(m - |I^0_{\exp}|)T = (m - |I^{0,1}_{\exp}(\sigma)|)T - |I^{0,2}_{\exp}(\sigma)|T \geq R$. Hence, the residual $m - |I^0_{\exp}|$ machines provide a processing time of at least R. So we can build a residual instance $I^\#$ for the residual $m^\# := m - |I^0_{\exp}|$ machines to place $J^\# := J(I^+_{\exp} \cup I^-_{\exp}) \cup \bigcup_{i \in I'_{\text{chp}}} C'_i$ with $C^\#_i := C_i$ for all expensive classes $i \in I^+_{\exp} \cup I^-_{\exp}$ as well as $C^\#_i := C'_i$ for all cheap classes $i \in I'_{\text{chp}}$. Apparently $I^\#$ is a nice instance and it actually holds the requirements of Theorem 10 (ii). In more detail we obtain that

$$\begin{aligned}
m^\# \cdot T &= (m - |I^0_{\exp}|)T \\
&\geq R = \sum_{i \in (I^+_{\exp} \cup I^-_{\exp})} (\lambda^\sigma_i s_i + P(C_i)) + \sum_{i \in I'_{\text{chp}}} (s_i + P(C'_i)) \\
&\geq P(J^\#) + \sum_{i \in I^+_{\exp}} \alpha'_i s_i + \sum_{i \in I^-_{\exp} \cup I'_{\text{chp}}} s_i \qquad \text{Lemma 5, } \alpha_i \geq \alpha'_i
\end{aligned}$$

and with Lemmas 5 and 6 and $\alpha_i \geq \alpha'_i, \alpha_i \geq 1$ we get that

$$\begin{aligned}
m^\# = m - |I^0_{\exp}| &\geq \sum_{i \in I_{\exp}} \alpha_i - |I^0_{\exp}| \\
&\geq \sum_{i \in I^+_{\exp}} \alpha'_i + |I^-_{\exp}| \geq \sum_{i \in I^+_{\exp}} \alpha'_i + \left\lceil \tfrac{1}{2} |I^-_{\exp}| \right\rceil.
\end{aligned}$$

So Theorem 10 leads us to use Algorithm 4 to complete our schedule feasibly with a makespan of at most $\frac{3}{2}T$. Figure 3.13 illustrates the use of Algorithm 4 with dashed lines around the area of the $m^\#$ last machines. □

3.7 OPEN QUESTIONS

There are several open questions. First of all: Is there a PTAS for the preemptive case? Remark that the splittable and preemptive case only differ in the (non)parallelization of jobs. However, the preemptive problem turns out to be much harder to approximate. Especially because Jansen et al. [67] have not found a PTAS by using n-fold ILPs, this remains as a very interesting open question. It might be an option to fix m (cf. Table 3.1 on page 35) for first results. Also, there may be constant bounds less than 3/2 with similar small running times.

Another promising direction might be the further investigation of unrelated/uniform machines; in fact, there is a constant approximation of Correa et al. [33] for the splittable case on unrelated machines and a PTAS for the non-preemptive case on uniform machines by Jansen et al. [71].

Also, we only discussed sequence-independent setups here. Considering sequence-*dependent* setups the setup times are given as a matrix $S \in \mathbb{Z}_{\geq 0}^{c \times c}$ of values $s_{(i_1,i_2)}$ which means that processing jobs of class i_2 on a machine currently set up for class i_1 will cost a setup time of $s_{(i_1,i_2)}$. For example there is a very natural application to TSP for $m = 1$ and $C_i = \{j_i\}$ with $t_{j_i} = 0$ where the jobs/classes identify cities. Selecting setups dependent on the previous job as well as the next job, we have the classical TSP (path version). There may be similar approximation results by (re)using the ideas of this chapter.

4

SCHEDULING WITH MANY SHARED RESOURCES

4.1 INTRODUCTION

We consider the problem of makespan minimization on identical parallel machines with many shared resources, or many shared resources scheduling (MSRS) for short. In this problem, we are given m identical machines, a set J of n jobs, and a processing time or size $p(j) \in \mathbb{N}_{\geq 0}$ for each job $j \in J$. Furthermore, each job needs exactly one additional shared resource in order to be executed and no other job needing the same resource can be processed at the same time. Hence, the jobs are partitioned into (non-empty) classes $\mathcal{C}$, i. e. $\bigcup \mathcal{C} = J$, such that each class corresponds to one of the resources. A schedule (σ, t) maps each job to a machine $\sigma : J \to \{1, \dots, m\}$ and a starting time $t : J \to \mathbb{N}_{\geq 0}$. It is called valid if no two jobs overlap on the same machine and no two jobs of the same class are processed in parallel. The makespan $C_{\max}$ of a schedule is defined as $\max_{j \in J} t(j) + p(j)$ and the goal is to find a schedule with minimum makespan. Note that MSRS also models the case in which some jobs do not need a resource since in this case private resources can be introduced.

STATE OF THE ART AND MOTIVATION The study of scheduling problems with additional resources has a long and rich tradition. Already in 1983, Blazewicz et al. [20] provided a classification for such problems along with basic hardness results and several additional surveys have been published since then [42, 19, 18]. The MSRS problem, in particular, was introduced by Hebrard et al. [61] who considered the scheduling of download plans for Earth observation satellites and provided a $(2m/(m+1))$-approximation for the problem. Strusevich [113] revisited MSRS and presented an additional application in human resource management. Moreover, he provided a faster, alternative $(2m/(m+1))$-approximation that is claimed to be simpler as well, and a 1.2-approximation for the case with only two machines. The work also extends the three-field problem classification for scheduling problems based on the convention for additional resources introduced in [20] to encompass the problem at hand. In particular, MSRS is denoted as $\mathrm{P}|\mathrm{res} \cdot 111|\mathrm{C}_{\max}$ in this notation. The most recent result regarding MSRS is due to Dósa et al. [40] who provided an EPTAS for MSRS with a constant number of machines. In fact, the EPTAS even works for a more general setting where each job j additionally may only be assigned to a machine belonging to a given set $\mathcal{M}(j)$ of eligible machines.

Since MSRS generalizes over $\mathrm{P}||\mathrm{C}_{\max}$, it is NP-hard already on two machines and strongly NP-hard for an arbitrary number of machines due to straightforward reductions from the PARTITION and 3-PARTITION problem, respectively. Hence, approximation schemes are essentially the best we can hope for.

The MSRS problem has also been considered with regard to the total completion time objective [78, 77]. The study of this variant is motivated by a scheduling problem in the semiconductor industry. On one hand, the authors show NP-hardness for generalizations of the problem, and on the other, they argue that the approach yielding a polynomial-time algorithm for total completion time minimization in the absence of resource constraints leads to a $(2 - 1/m)$-approximation for the considered problem.

Another way of looking at MSRS is to consider it as variant of scheduling with conflicts, where a conflict graph is given in which the jobs are the vertices and no two jobs connected by an edge may be processed at the same time. This problem was introduced for unit processing times by Baker and Coffman in 1996 [10]. It is known to be APX-hard [49] already on two machines with job sizes at most 4 and a bipartite agreement graph, i. e. the complement of the conflict graph. There are many positive and negative results for different versions of this problem (see, e.g., [10, 49] and the references therein). For instance, the problem is NP-hard on cographs with unit-size jobs but polynomial time solvable if the number of machines is constant [22]. Note that in the case of MSRS, we have a particularly simple cograph, i. e. a collection of disjoint cliques.

RESULTS We present a $1.\bar{6}$-approximation in Section 4.2, a 1.5-approximation in Section 4.3, and approximation schemes in Section 4.4. Note that the $1.\bar{6}$- and 1.5-approximation have better approximation ratios than the previously known $(2m/(m+1))$-approximation already for six and four machines, respectively.

The $1.\bar{6}$-approximation is a simple and fast algorithm that is based on placing full classes of jobs taking special care of classes containing jobs with particularly big sizes and of classes with large processing time overall. While the 1.5-approximation reuses some of the ideas and observations of the first result, it is much more involved. To achieve the second result, we first design a 1.5-approximation for the instances in which jobs cannot be too large relative to the optimal makespan and then design an algorithm that carefully places classes containing such large jobs and uses the first algorithm as a subroutine for the placement of the remaining classes. Note that our approaches are very different to the one in [61], which successively chooses jobs based on their size and the size of the remaining jobs in their class and then inserts them with some procedure designed to avoid resource conflicts, and the one in [113], which merges the classes into single jobs to avoid resource conflicts.

We provide an EPTAS for the variant of MSRS where the number of machines is constant and an EPTAS with resource augmentation for the general case. In particular, we need $\lfloor(1+\varepsilon)m\rfloor$ many machines in the latter result. Both results make use of the basic framework introduced in [67] which in turn utilizes relatively recent algorithmic results for IPs of a particular form – so-called n-fold IPs. Compared to the mentioned work by Dósa et al. [40] – which provides an EPTAS for the case with a constant number of machines as well – our result is arguably simpler and faster (going from at least triply exponential in m/ε to doubly exponential). We also provide the result with resource augmentation for the general case, which may be refined in the future to work without resource augmentation as well. Moreover, it seems

plausible that the use of n-fold IPs in the context of scheduling with additional resources may lead to further results in the future, which do not have to be limited to approximation schemes.

FURTHER RELATED WORK As mentioned above, there exists extensive research regarding scheduling with additional resources and we refer to the surveys [20, 42, 19, 18] for an overview. For instance, the variant with only one additional shared renewable resource where each job needs some fraction of the resource capacity has received a lot of attention (see [86, 73, 103, 72] for some relatively recent examples). Interestingly, Hebrard [61] pointed out that this basic setting is more closely related MSRS than it first appears: Consider the case that we have dedicated machines, i. e. each job is already assigned to a machine and we only have to choose the starting times, each job needs one unit of the singly additional shared resource, and the shared resource has some integer capacity. This problem is equivalent to MSRS if the multiple resources taken on the roles of the machines and the machines take the role of the single resource. Hence, results for variants of this setting translate to MSRS as well. For instance, MSRS can be solved in polynomial time if at most two classes include more than one job [84] and [56] yields a $(3+\varepsilon)$-approximation.

Scheduling with conflicts has also been studied from the orthogonal perspective, where jobs that are in conflict may not be processed on the same *machines*. This problem was already studied in the 1990's (see e. g. [23, 22]), and there has been a series of recent results [36, 53, 105] regarding the setting corresponding to MSRS where the conflict graph is a collection of disjoint cliques.

PRELIMINARIES We introduce some additional notation, and a first observation that will be used throughout the following sections.

For any set of jobs X let $p(X) = \sum_{j \in X} p(j)$ denote its total processing time. While creating or discussing a schedule, for any machine m denote by $p(m)$ the (current) total load of jobs on that machine m. Subsequently, for a set of machines M, $p(M) = \sum_{m \in M} p(m)$.

For any combination of a set $X \in \{J, \mathcal{C}\}$, a relation $* \in \{<, \leq, \geq, >\}$, and a number λ, we set $X_{*\lambda} = \{x \in X \mid p(x) * \lambda\}$. Similarly, given an interval v let $X_v = \{x \in X \mid p(x) \in v\}$. For example it holds that $J_{>1/2} = \{j \in J \mid p(j) > 1/2\}$ and also we have that $\mathcal{C}_{(1/2,3/4]} = \{c \in \mathcal{C} \mid p(c) \in (1/2, 3/4]\}$.

Note 1. *It holds that* $\mathrm{OPT} \geq \max\{\frac{1}{m}p(J), \max_{c \in \mathcal{C}} p(c)\}$.

Hence, we assume that $m < |\mathcal{C}|$ as otherwise there is a trivial schedule with one machine per class. Furthermore, let us assume that we sort the jobs in decreasing order of processing time. Consider the jobs j_m and j_{m+1} at position m and $m+1$. Note that it has to hold that $\mathrm{OPT} \geq p(j_m) + p(j_{m+1})$, since either j_{m+1} has to be scheduled on the same machine as one of the first m jobs, or two of the first m jobs have to be scheduled at the same machine.

4.2 A $1.\overline{6}$-APPROXIMATION

In this section we introduce a first simple algorithm that gives some intuition on the problem that will be used more cleverly in the next section. We start by lower bounding the makespan T of an optimal schedule and construct a schedule with makespan at most $\frac{5}{3}T$. The algorithm works by placing full classes of jobs in a specific order. More precisely, first classes that contain a job of size at least $\frac{1}{2}T$, then classes with total processing time larger than $\frac{2}{3}T$, and lastly all residual classes get placed.

Theorem 13. *Let $T = \max\{\frac{1}{m}p(J), \max_{c\in\mathcal{C}} p(c), p(j_m) + p(j_{m+1})\}$ where we write $J = \{j_1, \dots, j_n\}$ such that $p(j_1) \geq \cdots \geq p(j_n)$. For any instance of* MSRS *a schedule with makespan bounded by $\frac{5}{3}T$ can be computed in time $\mathcal{O}(|I|)$.*

As noted earlier, T denotes a lower bound on the makespan. We scale each job by $1/T$. As a consequence all jobs have a processing time in $(0, 1]$ and the total load is bounded by m. Denote by $\mathcal{C}_{B^+} = \{c \in \mathcal{C} : |c \cap J_{>1/2}| = 1\}$ all classes containing a job of size greater than $1/2$. We aim to find a schedule with makespan in $[1, 5/3]$. The following two observations are directly implied by the definition of T.

Observation 9. *For each class $c \in \mathcal{C}$ it holds that $|c \cap J_{>1/2}| \leq 1$.*

Observation 10. *It holds that $|\mathcal{C}_{B^+}| = |J_{>1/2}| \leq m$.*

Lastly, we address classes with a large total processing time.

Lemma 17. *Each class $c \in \mathcal{C}_{>2/3} \setminus \mathcal{C}_{B^+}$ can be partitioned into parts c_1 and $c_2 = c \setminus c_1$ such that $1/3 \leq p(c_1) \leq 2/3$ and $p(c_2) \leq 2/3$. This partition can be found in time $\mathcal{O}(|c|)$.*

Proof. If there exists a job $j_\top$ in c with $p(j_\top) > 1/3$, we define $c_1 = \{j_\top\}$ and $c_2 = c \setminus c_1$. Note that c does not contain a job with processing time larger than $1/2$ and hence, $p(c_1) \in (1/3, 1/2]$ and $p(c_2) = p(c) - p(c_1) < 1 - 1/3 = 2/3$.

Otherwise, greedily add jobs from c to an empty set c_1 until $p(c_1) \geq 1/3$ and set $c_2 = c \setminus c_1$. Since all the jobs of c have processing time at most $1/3$, it holds that $p(c_1) \in [1/3, 2/3]$. Consequently, it holds that $p(c_2) \leq 2/3$ as well. □

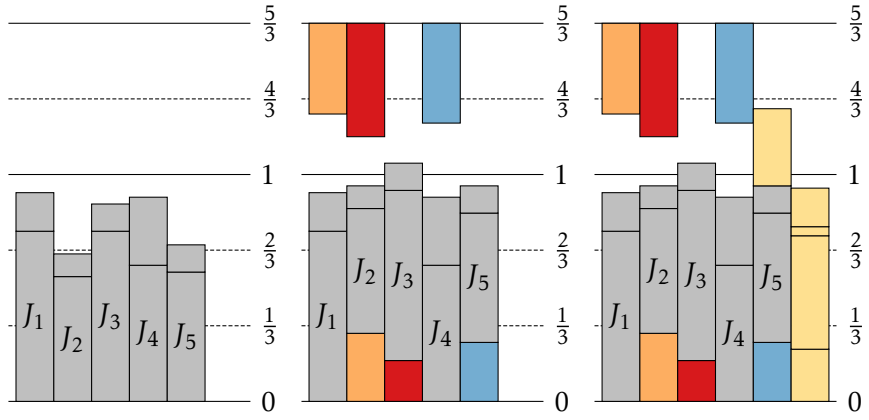
ALGORITHM FIVETHIRD

Step 1. Consider all classes containing a job with processing time larger than $1/2$, $\mathcal{C}_{B^+}$. Each of these classes is assigned to an individual machine, and all jobs from such a class are scheduled consecutively, see Fig. 4.1a.

Step 2. Consider all remaining classes with total processing time larger than $2/3$, $\mathcal{C}_{>2/3} \setminus \mathcal{C}_{B^+}$. Try to add these classes on the machines filled with the classes $\mathcal{C}_{B^+}$ and afterward proceeds to empty machines, see Fig. 4.1b. If the considered machine has load in $(1, 5/3]$, *close* the machine and no longer attempt to place any other job on it. Note that after placing the classes $\mathcal{C}_{B^+}$ all machines remained open. Let m_i be the machine we try to place class $c \in \mathcal{C}_{>2/3} \setminus \mathcal{C}_{B^+}$ on. If m_i has load $p(m_i) \leq 5/3 - p(c)$, place the entire class on this machine and close it. Otherwise, partition the class c in two parts c_1 and

c_2 such that $p(c_2) \leq p(c_1) \leq 2/3$ (cf. Lemma 17). Place the larger part c_1 on the current machine starting at $5/3 - p(c_1)$ and close it, moving to the next machine. All jobs on this machine are delayed such that the first job starts at $p(c_2)$. All jobs from c_2 are scheduled between 0 and $p(c_2)$ on this machine. If it has load of at least 1, this machine is closed as well.

Step 3 (Greedy). Finally, place the classes $\mathcal{C}_{\leq 2/3} \setminus \mathcal{C}_{B^+}$, see Fig. 4.1c. Consider the residual machines one after another and add each class $c \in \mathcal{C}_{\leq 2/3} \setminus \mathcal{C}_{B^+}$ entirely to the considered machine. As soon as the load of a machine exceeds 1 close it and move to the next.



(a) Classes with large jobs (b) Placing large classes (c) Adding all other classes

Figure 4.1: The three steps of the algorithm (where $J_{>1/2} = \{J_1, \ldots, J_5\}$)

ANALYSIS

Lemma 18. *Given any instance $I = (m, \mathcal{C})$ of* MSRS, *FiveThird produces a feasible schedule with makespan at most $\frac{5}{3}\,\mathrm{OPT}(I)$.*

Proof. We have to prove that all jobs can be scheduled, the processing times of two jobs from the same class never overlap, and the latest completion time of a job is at most 5/3.

We start by proving that all jobs are scheduled, by showing that the algorithm closes only machines that have a total load of at least 1. Since the total load of the jobs is bounded by m, when attempting to schedule the last class, there has to exist a non closed machine. The only time the algorithm potentially closes a machine with load less than 1 is in step 2 when a class $\mathcal{C}_{>2/3} \setminus \mathcal{C}_{B^+}$ is split into two parts. Let c_{B^+} be the class already on the machine and c_1 and c_2 be the parts of the class the algorithm tries to schedule in this step, such that $p(c_1) \geq p(c_2)$. Since the class was split in two by the algorithm it holds that $p(c_{B^+}) + p(c_1) + p(c_2) > 5/3$. Furthermore, since $p(c_1) + p(c_2) \leq 1$ and $p(c_1) \geq p(c_2)$ it holds that $p(c_2) \leq 1/2$ and hence $p(c_{B^+}) + p(c_1) > 7/6$. Hence that closed machine has a load of at least 1.

Next, we prove that the processing of two jobs from the same class never overlaps in time. Again, the only time one class is scheduled on more than one machine is step 2. When placing the two parts these parts do not overlap, since they have a processing time of at most 1 and one of the parts starts at 0

while the other ends at 5/3. The algorithm does not generate any overlapping by shifting jobs already on the machine, since those have to originate from classes in $\mathcal{C}_{B^+}$, which each got placed on an individual machine.

Finally, we prove that the latest completion time of a job is at most 5/3. After step 1 all the machines have a load of at most 1, since each class has a total processing time of at most 1. In step 2, we only add an entire class if the total load is bounded by 5/3. If a class is split, the part that is added has a total processing time of at most 2/3. Since before adding this part the machine had a load of at most 1, the load of the closed machine is bounded by 5/3. This concludes the proof of Theorem 13. □

4.3 A 1.5-APPROXIMATION

In this section we introduce the more involved algorithm hinted at earlier. While the general idea is similar, finding a lower bound T for the makespan and then placing classes depending on included big jobs and total processing time, the steps are a lot more granular. We first give a 1.5-approximation algorithm for instances without jobs of size bigger than $3/4T$. After that we introduce a second 1.5-approximation algorithm that places classes with jobs of size bigger than $3/4T$ on distinct machines and fills them with other jobs in a clever way such that we can reuse the first algorithm for the remaining classes.

Theorem 14. *There exists an algorithm that for any given instance I of* MSRS *finds a schedule with makespan bounded by $\frac{3}{2}\,\mathrm{OPT}$ in $\mathcal{O}(n + m\log(m))$ steps.*

In the following let us assume that we have scaled the instance such that $\mathrm{OPT} = 1$. In order to provide a 1.5-approximation algorithm, we split the jobs of the instance into *huge* jobs $J_H = J_{>3/4} = \{j \in J \mid p(j) > 3/4\}$, *big* jobs $J_B = J_{(1/2,3/4]} = \{j \in J \mid p(j) \in (1/2, 3/4]\}$, *medium* jobs $J_M = J_{(1/4,1/2]} = \{j \in J \mid p(j) \in (1/4, 1/2]\}$, and all residual jobs (with a processing time of at most 1/4) which we refer to as *small* jobs. Furthermore, turning to the classes $\mathcal{C}$ we define the subset $\mathcal{C}_H = \{c \in \mathcal{C} : |J_H \cap c| = 1\}$ of all classes containing a huge job, the subset $\mathcal{C}_B = \{c \in \mathcal{C} : |J_B \cap c| = 1\}$ of all classes containing a big job, the subset $\mathcal{C}_{\geq 3/4} = \{c \in \mathcal{C} \mid p(c) \geq 3/4\}$ of all classes with a total processing time of at least 3/4, and the subset $\mathcal{C}_{(1/2,3/4)} = \{c \in \mathcal{C} \mid p(c) \in (1/2, 3/4)\}$ of all classes with a total processing time in $(1/2, 3/4)$.

Lemma 19. *For any normalized optimal schedule and the corresponding partition of $\mathcal{C}$ into $\mathcal{C}_H, \mathcal{C}_B, \mathcal{C}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)$ and $\mathcal{C} \setminus \mathcal{C}_{\geq 3/4}$ it holds that*

$$|\mathcal{C}_H| + \max\left\{|\mathcal{C}_B|, \left\lceil \frac{1}{2}\left(|\mathcal{C}_B| + |\mathcal{C}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)|\right)\right\rceil\right\} \leq m.$$

Proof. Clearly, it holds that $|\mathcal{C}_H| + |\mathcal{C}_B| \leq m$. Let us consider the total load processed in the time corridor between 1/4 and 3/4 (over the entire schedule). For each class $c \in \mathcal{C}_H$ we have to schedule at least load 1/2 in this corridor, since the tallest job in c, which has a processing time of at least 3/4, has to start before 1/4 and has to end after 3/4. For each class $c \in \mathcal{C}_B$, at least load 1/4 is scheduled in this corridor since its big job, which has a processing time in $(1/2, 3/4)$, has to end after 1/2 and has to start before 1/2. Finally,

each class in $\mathcal{C}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)$ has load of at least 3/4. Since at most 1/2 of this load can be scheduled outside of the corridor, there has to be load of at least 1/4 scheduled inside of this corridor. Hence the total load scheduled in this corridor is at least $\frac{1}{2}|\mathcal{C}_H| + \frac{1}{4}(|\mathcal{C}_B| + |\mathcal{C}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)|)$.

Since each machine covers at most a half of this load, it holds that

$$\begin{aligned} m &\geq \left\lceil \frac{\frac{1}{2}|\mathcal{C}_H| + \frac{1}{4}(|\mathcal{C}_B| + |\mathcal{C}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)|}{\frac{1}{2}} \right\rceil \\ &= |\mathcal{C}_H| + \left\lceil \frac{1}{2}(|\mathcal{C}_B| + |\mathcal{C}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)|) \right\rceil \end{aligned}$$

and that proves the claim. □

Next, we prove that in $\mathcal{O}(n + m\log(m))$ steps it is possible to find the smallest value T with $\max\{\frac{1}{m}p(J), \max_{c\in\mathcal{C}} p(c)\} \leq T \leq \mathrm{OPT}$ such that the instance scaled by $1/T$ fulfills the properties from Observations 9 and 10 and Lemma 19. The algorithms presented in this section will find a schedule with makespan at most 3/2 for this scaled instance, i. e. the schedule for the original instance will have a makespan of at most $(3/2)T$.

Lemma 20. *In time $\mathcal{O}(n + m\log(m))$ for any given instance I, it is possible to find a lower bound $T \leq \mathrm{OPT}$ such that for the instance normalized by $1/T$ and the corresponding partition of $\mathcal{C}$ into $\mathcal{C}_H, \mathcal{C}_B, \mathcal{C}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)$ and $\mathcal{C} \setminus \mathcal{C}_{\geq 3/4}$ it holds that*

$$|\mathcal{C}_H| + \max\left\{ |\mathcal{C}_B|, \left\lceil \frac{1}{2}(|\mathcal{C}_B| + |\mathcal{C}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)|) \right\rceil \right\} \leq m.$$

Proof. By Note 1, we know that we can set $T \geq \max\{\frac{1}{m}p(J), \max_{c\in\mathcal{C}} p(c)\}$. Let $\tilde{p}_i$ denote the i-st largest processing time (in a list of processing times $\tilde{p}_1 \geq \cdots \geq \tilde{p}_n$ containing one entry per job). Since each machine can contain at most one job with processing times larger than $\mathrm{OPT}/2$, we set $T \geq \max\{\frac{1}{m}p(J), \max_{c\in\mathcal{C}} p(c), \tilde{p}_m + \tilde{p}_{m+1}\}$. It is possible to find a particular $\tilde{p}_i$ in time $\mathcal{O}(n)$, by using the famous median algorithm of Blum et al. [21].

Since each class in $\mathcal{C}_H \cup \mathcal{C}_B$ contains an item with processing time $\geq 1/2$, only the m classes containing the largest items are candidates for these sets. These classes can be found in $\mathcal{O}(n)$ by identifying the largest item of each class and comparing it to p_{m+1}. Similarly the number of classes in $\mathcal{C}_{\geq 3/4}$ is bounded by $(4/3)m$, which can be identified in $\mathcal{O}(n)$ by comparing their processing time to $\max\{\frac{1}{m}p(J), \max_{c\in\mathcal{C}} p(c), \tilde{p}_m + \tilde{p}_{m+1}\}$.

After identifying the potential classes, we have to deal with at most $\mathcal{O}(m)$ classes. For each of these classes there exist three threshold values for $T \in \mathbb{N}$ (i. e. $\lceil \frac{4}{3}(\max_{j\in c} p(j)) + 1/3 \rceil$, $2(\max_{j\in c} p(j)) + 1$, and $\lceil \frac{4}{3}p(c) + 1/3 \rceil$), that would categorize these classes to be no longer in $\mathcal{C}_H$, $\mathcal{C}_B$, and $\mathcal{C}_{\geq 3/4}$, respectively, which after the first two steps can be found in $\mathcal{O}(m)$ for all the classes, since they depend on the largest processing time in the class and the total processing time of that class.

The algorithm can take all these values and sort them by size in time $\mathcal{O}(m\log(m))$. Via binary search in time $\mathcal{O}(m\log(m))$, it is possible to find the smallest value T such that $T \geq \max\{\frac{1}{m}p(J), \max_{c\in\mathcal{C}} p(c), 2p_{m+1}\}$ and for

the instance normalized by $1/T$ and the corresponding partition into of $\mathcal{C}$ into $\mathcal{C}_H, \mathcal{C}_B, \mathcal{C}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)$ and $\mathcal{C} \setminus \mathcal{C}_{\geq 3/4}$ it holds that

$$|\mathcal{C}_H| + \max\left\{ |\mathcal{C}_B|, \left\lceil \frac{1}{2} \left(|\mathcal{C}_B| + |\mathcal{C}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)| \right) \right\rceil \right\} \leq m. \qquad \square$$

In the following, we only consider the instance that was scaled by $1/T$. We present two Lemmas stating the possibility to partition some classes into two parts that will be scheduled on two different machines.

Lemma 21. *Let $c \in \mathcal{C}_{\geq 3/4}$ and $\max_{j \in c} p(j) \leq 3/4$. Then c can be partitioned into two parts $\check{c}$ and $\hat{c}$ with $p(\check{c}) \leq 1/2$ and $p(\hat{c}) \leq 3/4$ and $p(\check{c}) \leq p(\hat{c})$. Furthermore, if $\max_{j \in c} p(j) \leq 1/2$, it holds that $p(\check{c}) \in (1/4, 1/2]$ or $p(\hat{c}) \in (1/4, 1/2]$.*

Proof. Let $c \in \mathcal{C}_{\geq 3/4}$ and $\max_{j \in c} p(j) \leq 3/4$. If $\max_{j \in c} p(j) > 1/2$, we set $\hat{c}$ to include the job from c with size bigger than $1/2$ and $\check{c} = c \setminus \hat{c}$. If it holds that $\max_{j \in c} p(j) \in (1/4, 1/2]$, then let c' include a maximal job from c and let $\hat{c}, \check{c} \in \{c', c \setminus c'\}$ be distinct such that $p(\check{c}) \leq p(\hat{c})$. Lastly, if $\max_{j \in c} p(j) \leq 1/4$, then we construct c' by greedily adding jobs from c to c' until $p(c') > 1/4$ and again define $\hat{c}, \check{c} \in \{c', c \setminus c'\}$ to be distinct such that $p(\check{c}) \leq p(\hat{c})$. $\square$

Lemma 22. *Let $c \in \mathcal{C}$ with $p(c) \in (1/2, 3/4)$ and $\max_{j \in c} p(j) \leq 1/2$. Then c can be partitioned into two parts $\check{c}$ and $\hat{c}$ with $p(\check{c}) \leq p(\hat{c}) \leq 1/2$ and $1/4 < p(\hat{c})$.*

Proof. Let $c \in \mathcal{C}$ with $p(c) \in (1/2, 3/4)$ and $\max_{j \in c} p(j) \leq 1/2$. Now, if we have $\max_{j \in c} p(j) \in (1/4, 1/2]$, then let c' include a maximal job from c and let $\hat{c}, \check{c} \in \{c', c \setminus c'\}$ be distinct such that $p(\check{c}) \leq p(\hat{c})$.

If $\max_{j \in c} p(j) \leq 1/4$, then we construct c' by greedily adding jobs from c to c' until $p(c') > 1/4$ and again define $\hat{c}, \check{c} \in \{c', c \setminus c'\}$ to be distinct such that $p(\check{c}) \leq p(\hat{c})$. $1/4 < p(\hat{c})$ follows directly from the fact that $p(\check{c}) \leq p(\hat{c})$ and $1/2 < p(\check{c}) + p(\hat{c})$. $\square$

In the following, we will present two algorithms. The first can only handle instances with classes that do not possess an item with processing time larger than 3/4. This algorithm will be used as a subroutine for the second algorithm, which can handle all instances.

4.3.1 *Instances without Huge Jobs*

Here we give an algorithm for instances with $|\mathcal{C}_H| = 0$. We assume that the instance was scaled by a value $1/T$ and the classes are categorized as described earlier. The main idea is to repeatedly take combinations of classes with specific parameters which conveniently fill one, two or three machines, without opening additional ones. *Fill* in this case means that the average load of *full* machines is in $[1, 3/2]$. We start with taking two classes with total size in $(1/2, 3/4)$ each, as those fill one machine. Then we continue with four classes with total size $\geq 3/4$ each, and show how those can be arranged to fill three machines. The procedure continues with different combinations of classes until all jobs are scheduled. We show the correctness of the algorithm by arguing that closed machines have on average load of at least 1, and every scheduled jobs is finished at 3/2. At some point in the algorithm we reach a

state where only jobs of classes with total load at most 1/2 are left. Those can be scheduled greedily, by placing full classes on residual machines, until a machine has load at least 1.

Since we repeatedly have to refer to the jobs which have not been scheduled, we introduce the notation of $\bar{\mathcal{C}}_X \subseteq \mathcal{C}_X$ to denote the subset of classes that have not been scheduled at the described step for any class specifier X. Note that in the beginning of the algorithm, we have $\bar{\mathcal{C}}_X = \mathcal{C}_X$ for all the sets. Furthermore, the algorithm will close some of the machines during the construction of the schedule and will not add jobs to closed machines. We denote the set of closed machines as M_c. The algorithm is as follows:

ALGORITHM NOHUGE

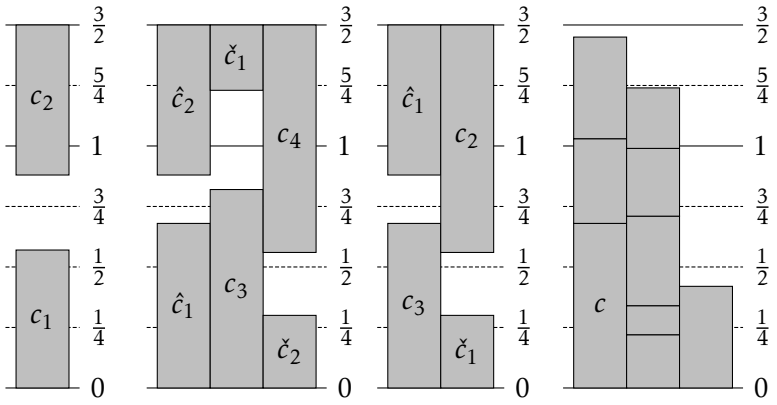
Step 1. By applying Lemma 21, partition every class $c \in \mathcal{C}_{>3/4}$ into two parts $\check{c}, \hat{c} \subseteq c$ with $p(\check{c}) \leq p(\hat{c}) \leq 3/4$ and $p(\check{c}) \leq 1/2$.

Step 2. While $|\bar{\mathcal{C}}_{(1/2,3/4)}| \geq 2$: Take $c_1, c_2 \in \bar{\mathcal{C}}_{(1/2,3/4)}$. Schedule c_1 and c_2 on one machine such that c_1 starts at 0 and c_2 ends at 3/2.

Claim. *The load of each machine closed in this step is in* $(1, 3/2)$. *After this step it holds that* $|\bar{\mathcal{C}}_{(1/2,3/4)}| \leq 1$, *the partial schedule is feasible, and the total load of closed machines* M_c *is at least* $|M_c|$.

Step 3. While $|\bar{\mathcal{C}}_{\geq 3/4}| \geq 4$: Take $c_1, c_2, c_3, c_4 \in \bar{\mathcal{C}}_{\geq 3/4}$. On the first machine schedule $\hat{c}_1$ and $\hat{c}_2$, such that $\hat{c}_1$ starts at 0 and $\hat{c}_2$ ends at 3/2. On the second machine schedule $\check{c}_1$ and c_3, such that $\check{c}_1$ ends at 3/2 and starts after 1. On the third machine schedule $\check{c}_2$ and c_4, such that $\check{c}_2$ starts at 0 and ends before 1/2 followed by c_4, see Fig. 4.2b for an example. Close all three machines.

Claim. *After this step* $|\bar{\mathcal{C}}_{(1/2,3/4)}| \leq 1$ *and* $|\bar{\mathcal{C}}_{\geq 3/4}| \leq 3$, *the partial schedule is feasible, and the total load of closed machines* M_c *is at least* $|M_c|$. *Furthermore, all scheduled jobs are finished by* 3/2.



(a) Step 2 (b) Step 3 (c) Step 4 (d) Step 5

Figure 4.2: Examples for Steps 2 to 5

Step 4. If $|\bar{\mathcal{C}}_{\geq 3/4}| \geq 2$ and $|\bar{\mathcal{C}}_{(1/2,3/4)}| = 1$: Take $c_1, c_2 \in \bar{\mathcal{C}}_{\geq 3/4}$ and $c_3 \in \bar{\mathcal{C}}_{(1/2,3/4)}$. Schedule c_3 on the first machine, followed by $\hat{c}_1$ such that it ends at 3/2. Schedule $\check{c}_1$ on the second machine followed by the jobs from c_2 and close both machines, see Fig. 4.2c for an example.

Claim. *After this step it holds that* $|\overline{\mathcal{C}}_{(1/2,3/4)}| = 0$ *and* $|\overline{\mathcal{C}}_{\geq 3/4}| \leq 3$ *or it holds that* $|\overline{\mathcal{C}}_{(1/2,3/4)}| = 1$ *and* $|\overline{\mathcal{C}}_{\geq 3/4}| \leq 1$*. This implies that* $|\overline{\mathcal{C}}_{>1/2}| \leq 3$ *after this step and that* $\overline{\mathcal{C}}_{>1/2}$ *contains at most one class with total processing time less than* $3/4$*. Furthermore, the partial schedule is feasible, the total load of closed machines* M_c *is at least* $|M_c|$*, and no scheduled job finishes after* $3/2$*.*

Depending on the size of $|\overline{\mathcal{C}}_{>1/2}|$ the algorithm chooses one of three procedures:

Step 5. If $|\overline{\mathcal{C}}_{>1/2}| \leq 1$: Place this class c on one machine. Fill this machine and the residual machines greedily with the residual classes in $\overline{\mathcal{C}}_{\leq 1/2}$.

Claim. *After this step it either holds that* $2 \leq |\overline{\mathcal{C}}_{>1/2}| \leq 3$ *or all jobs have been scheduled feasibly with no job finishing after* $3/2$*.*

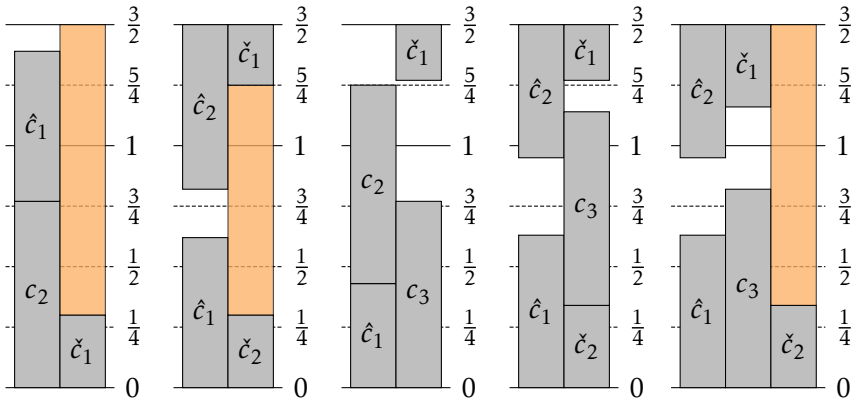
Proof. In the latter case, we can place all remaining jobs, since there are at least as many open machines as there is open load because before this step we had $p(M_c) \geq |M_c|$. Each opened machine will be filled with load in $[1, 3/2]$, since each residual job has a size of at most $1/2$. □

Step 6. If $|\overline{\mathcal{C}}_{>1/2}| = 2$: Let $\overline{\mathcal{C}}_{>1/2} = \{c_1, c_2\}$ with $p(c_1) \geq p(c_2)$. We know that $p(c_1) \geq 3/4$.

1. If $p(c_2) \leq 3/4$:
   a) If $p(c_1) + p(c_2) \leq 3/2$: Schedule both on one machine (with c_1 starting at 0 and c_2 ending at $3/2$), close it, and continue greedily with the residual jobs.
   b) If $p(c_1) + p(c_2) > 3/2$: Place c_2 on one machine followed by $\hat{c}_1$ such that $\hat{c}_1$ ends at $3/2$ and close the machine. Place $\check{c}_1$ on the next machine and continue greedily with the residual jobs in $\overline{\mathcal{C}}_{\leq 1/2}$.
2. If $p(c_2) \geq 3/4$:
   a) If $p(\hat{c}_1) + p(\hat{c}_2) \leq 1$: Schedule c_2 followed by $\hat{c}_1$ on one machine and close it. Start $\check{c}_1$ at 0 on the next machine and continue greedily with the residual jobs in $\overline{\mathcal{C}}_{\leq 1/2}$.
   b) If $p(\hat{c}_1) + p(\hat{c}_2) > 1$: Then place $\hat{c}_1$ and $\hat{c}_2$ on one machine such that $\hat{c}_1$ starts at 0 and $\hat{c}_2$ ends at $3/2$. Place $\check{c}_2$ at the bottom and $\check{c}_1$ at the top of the next machine. Continue greedily with the residual classes in $\overline{\mathcal{C}}_{\leq 1/2}$. Start placing them between $\check{c}_2$ and $\check{c}_1$ until the load of that machine is at least 1 and then continue with the empty machines.

Claim. *After this step it either holds that* $|\overline{\mathcal{C}}_{>1/2}| = 3$ *or all jobs have been scheduled feasibly with no job finishing after* $3/2$*.*

Proof. We will prove the latter case. If $p(c_2) \leq 3/4$, the load of the machines that contains either c_1 and c_2 or only c_2 and $\hat{c}_1$ has a load in $(1, 3/2]$. Each residual class (or part of a class) has a total processing time of at most $1/2$. Furthermore, up to this step, it holds that $p(M_c) \geq |M_c|$. As a consequence, greedily scheduling the residual classes starting with $\check{c}_1$ is possible.



(a) Step 6.2a (b) Step 6.2b (c) Step 7.1 (d) Step 7.2a (e) Step 7.2b

Figure 4.3: Examples for Step 6 and Step 7. Orange blocks represent space for residual classes.

If, on the other hand, $p(c_2) > 3/4$ holds, the machine containing c_2 and $\hat{c}_1$ (or $\hat{c}_2$ and $\hat{c}_1$ respectively) has a total load of at least 1 in either case, and placing $\check{c}_1$ (and $\check{c}_2$) as described does not provoke an overlapping of two jobs requiring the same resource (see Fig. 4.3). Furthermore, the machine containing c_2 and $\hat{c}_1$ (or $\hat{c}_2$ and $\hat{c}_1$ respectively) has a total load of at most 3/2 since $p(\check{c}_2) + p(\hat{c}_1) + p(\hat{c}_2) \leq 1/2 + 1$ if $p(\hat{c}_1) + p(\hat{c}_2) \leq 1$ and $p(\hat{c}_i) \leq 3/4$ for $i \in \{1, 2\}$. The residual classes again can be scheduled greedily. This is easy to see in the case $p(\hat{c}_1) + p(\hat{c}_2) \leq 1$ and otherwise we have $p(\check{c}_2) + p(\check{c}_1) \in [0, 1)$ and hence the remaining gap has a size of at least 1/2. Since all remaining classes have total load of at most 1/2 it is possible to greedily add such classes until the total load of that machine is at least 1 or all remaining classes have been placed. □

Step 7. If $|\bar{\mathcal{C}}_{>1/2}| = 3$: Then $\bar{\mathcal{C}}_{>1/2} = \bar{\mathcal{C}}_{\geq 3/4}$. Let $\bar{\mathcal{C}}_{\geq 3/4} = \{c_1, c_2, c_3\}$.

1. If there exists an $i \in \{1, 2, 3\}$ such that $\hat{c}_i \leq 1/2$: Let w.l.o.g. $\hat{c}_1 \leq 1/2$. On the first machine schedule $\hat{c}_1$ followed by all the jobs from c_2. On the next machine schedule all the jobs from c_3 and the job $\check{c}_1$ such that it ends at 3/2 and close both machines. Greedily schedule the jobs in $\bar{\mathcal{C}}_{\leq 1/2}$ on the non-closed machines.

2. If $\hat{c}_i > 1/2$ for all $i \in \{1, 2, 3\}$: Place $\hat{c}_1$ and $\hat{c}_2$ on one machine such that $\hat{c}_1$ starts at 0 and $\hat{c}_2$ ends at 3/2.

   a) If $p(\check{c}_1) + p(\check{c}_2) + p(c_3) \leq 3/2$: On the next machine place $\check{c}_2$ followed by c_3 and $\check{c}_1$ and let $\check{c}_1$ end at 3/2. Close both machines.

   b) If $p(\check{c}_1) + p(\check{c}_2) + p(c_3) > 3/2$: Then w.l.o.g. $p(\check{c}_1) > 1/4$ and we place c_3 and $\check{c}_1$ on the next machine, such that $\check{c}_1$ ends at 3/2. Close both machines. On the next machine place $\check{c}_2$ such that it starts at 0.

   Greedily schedule the jobs in $\bar{\mathcal{C}}_{\leq 1/2}$ on the non-closed machines.

Claim. *After this step, all scheduled jobs are finished by* 3/2 *and the schedule is feasible.*

Proof. Note that the two machines containing the classes c_1, c_2, and c_3 (or c_1, $\hat{c}_2$, and c_3 respectively) have a total load of at least 2. As a consequence, all machines M_c closed to this point have a load of at least $|M_c|$. As a consequence there residual load fits on the residual machines. When greedily scheduling the classes each machine is overloaded by at most $1/2$, since each residual class has a processing time of at most $1/2$. □

Lemma 23. *Given an instance $I = (m, J, \mathcal{C})$ that does not contain a huge job, the algorithm NoHuge finds a schedule with makespan at most $\frac{3}{2}T$, where $T = \max\{\frac{1}{m}p(J), \max_{c\in\mathcal{C}} p(c), \tilde{p}_m + \tilde{p}_{m+1}\}$.*

4.3.2 *General Instances*

Now we present the above-mentioned algorithm that can handle any instance of the problem and uses the previous algorithm in a subroutine. More specifically, this algorithm places all classes which contain a huge job on a separate machine and fills those machines with jobs from other classes. This is done by working through different combinations of classes until we reach a point where we can handle the remaining classes and machines as a separate problem instance, at which point the previous algorithm is used. As before we assume that the instance is scaled by a value $1/T$ and the classes are categorized as described earlier.

We keep the following invariant of the remaining instance over the whole algorithm.

Invariant. The total load of unscheduled jobs and jobs placed on open machines is bounded by the number of open machines (open machines are all machines not explicitly closed) and in each step the cardinality of the set of unused machines $\overline{M}_u$ is at least

$$|\overline{M}_u| \geq \max\{|\bar{\mathcal{C}}_B|, \lceil(|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)|)/2\rceil\}.$$

ALGORITHM THREEHALF

Step 1. Combine specific jobs of the same class into one job. The simplification is done as follows: Iterate all classes $c \in \mathcal{C}$

- If $c \in \mathcal{C}_H$ combine all jobs in c to one huge job.
- Else if $p(c) > 3/4$ partition it into parts $\hat{c}$ and $\check{c}$ with $p(\check{c}) \leq p(\hat{c}) \leq 3/4$ and $p(\check{c}) \leq 1/2$. Introduce for each part a new job with processing time $p(\hat{c})$ and $p(\check{c})$, see Lemma 21.
- Else if $c \in \mathcal{C}_{(1/2,3/4)} \cap \mathcal{C}_B$: partition it into $\hat{c}$ and $\check{c}$, such that $\hat{c}$ contains the largest job and $\check{c}$ contains the rest.
- Else if $c \in \mathcal{C}_{(1/2,3/4)} \setminus \mathcal{C}_B$ partition it into $\hat{c}$ and $\check{c}$ with $p(\check{c}) \leq p(\hat{c}) \leq 1/2$, see Lemma 22.
- Else if $p(c) \leq 1/2$ introduce one job of size $p(c)$.

Claim. *This partition is feasible and every solution for this simplified instance, will still be a solution for the original instance.*

Step 2. For each $c \in \mathcal{C}_H$: Open one new machine and assign class c to it. Let M_H be the set of these opened machines. Close all the machines that have load exactly 1. Denote by $\overline{M}_H$ the set of currently open machines containing a class from $\mathcal{C}_H$.

Claim. *After this step, there are $|\overline{M}_H|$ many open machines with load in $(3/4, 1)$, $|\bar{\mathcal{C}}_H| = 0$. For the residual empty machines $\overline{M}_u$ it holds that*

$$|\overline{M}_u| \geq \max\{|\bar{\mathcal{C}}_B|, \lceil(|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus (\mathcal{C}_H \cup \mathcal{C}_B)|)/2\rceil\}$$

and

$$p(\overline{M}_H) + p(\bar{\mathcal{C}}) \leq |\overline{M}_u| + |\overline{M}_H|.$$

Step 3. Assign classes c_s with $p(c_s) \leq 1/2$ greedily to machines $\overline{M}_H$ and close each machine with load at least 1. Continue until either no machines in $\overline{M}_H$ with load less than 1 is left, or no class with load at most $1/2$ is left. If $|\overline{M}_H| = 0$, continue with NoHuge on the residual instance.

Claim. *After this step either all jobs are scheduled feasibly or it holds that $|\overline{M}_H| \geq 1$ and $|\bar{\mathcal{C}}_{\leq 1/2}| = 0$. Furthermore, the partial schedule is feasible, all scheduled jobs are finished by $3/2$ and for the residual empty machines $\overline{M}_u$ it holds that*

$$|\overline{M}_u| \geq \max\{|\bar{\mathcal{C}}_B|, \lceil(|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|)/2\rceil\}$$

and

$$p(\overline{M}_H) + p(\bar{\mathcal{C}}) \leq |\overline{M}_u| + |\overline{M}_H|.$$

Proof. Since we only close machines with load at least one in this step and did not open any new machine, the invariant on the number of unused machines is trivially true. Hence, if we have used NoHuge on the residual instance, by Lemma 23 it generates a schedule with makespan at most $3/2$ because $p(\bar{\mathcal{C}}) \leq |\overline{M}_u|$ at that point and no class was scheduled partially. □

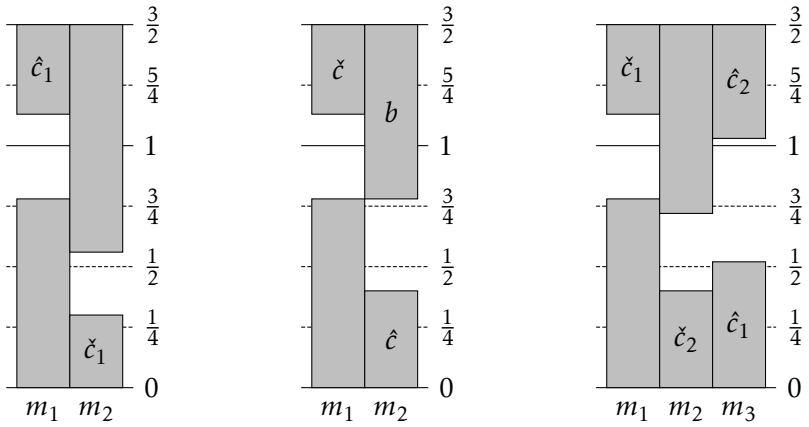
Step 4. While $|\overline{M}_H| \geq 2$ and $|\bar{\mathcal{C}}_{(1/2,3/4)} \setminus \bar{\mathcal{C}}_B| \geq 1$: Take $m_1, m_2 \in \overline{M}_H$, $c \in \bar{\mathcal{C}}_{(1/2,3/4)} \setminus \mathcal{C}_B$. Shift the huge job on m_2 up such that it ends at $3/2$ and starts at or after $1/2$. Schedule $\hat{c}$ on m_1 such that it ends at $3/2$, schedule $\check{c}$ on m_2 starting at 0 and close both machines, see Fig. 4.4a. If $|\overline{M}_H| = 0$, continue with NoHuge on the residual instance.

Claim. *After this step either all jobs are scheduled feasibly or one of the following two conditions holds: $|\overline{M}_H| = 1$ and $|\bar{\mathcal{C}}_{\leq 1/2}| = 0$, or $|\overline{M}_H| \geq 2$ and $|\bar{\mathcal{C}} \setminus (\mathcal{C}_B \cup \bar{\mathcal{C}}_{\geq 3/4})| = 0$. Furthermore, the partial schedule is feasible, all scheduled jobs are finished by $3/2$ and all machines not in $\overline{M}_H$ are either closed or empty and for the residual empty machines $\overline{M}_u$ it holds that*

$$|\overline{M}_u| \geq \max\{|\bar{\mathcal{C}}_B|, \lceil(|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|)/2\rceil\}$$

and

$$p(\overline{M}_H) + p(\bar{\mathcal{C}}) \leq |\overline{M}_u| + |\overline{M}_H|.$$



(a) Step 4 (b) Step 6 (c) Step 8

Figure 4.4: Examples for Step 4, Step 6, and Step 8

Proof. Note that we have not opened any other machine in this step, hence the lower bound on $|\overline{M}_u|$ is trivially true. The total load of m_1, m_2 and c is at least $2 \cdot 3/4 + 1/2 = 2$. Hence in each of these steps, we close two machines but also reduce the residual load by at least 2, proving the upper bound on the residual load. Hence, if we have used NoHuge on the residual instance, by Lemma 23 it generates a schedule with makespan at most 3/2 because $p(\bar{\mathcal{C}}) \leq |\overline{M}_u|$ at that point and no class was scheduled partially. □

Step 5. If $|M_H| = 1$:

- If there exists $c \in \bar{\mathcal{C}} \setminus \mathcal{C}_B$: Choose $c' \in \{\hat{c}, \check{c}\}$ with $c' \in (1/4, 1/2]$. Schedule c' on the last open machine m_0. Use NoHuge to schedule the residual instance, including the job $c'' \in c \setminus c'$. "Rotate" the load on m_0, such that c' does not overlap with c''.
- If $\bar{\mathcal{C}} \setminus \mathcal{C}_B$ is empty: Assign all the residual classes to an individual machine.

Claim. *After this step all jobs have been scheduled feasibly or* $|\overline{M}_H| \geq 2$ *and* $|\bar{\mathcal{C}} \setminus (\mathcal{C}_B \cup \bar{\mathcal{C}}_{\geq 3/4})| = 0$. *Additionally the partial schedule is feasible, all scheduled jobs are finished by* 3/2 *and for the residual empty machines* $\overline{M}_u$ *it holds that*

$$|\overline{M}_u| \geq \max\{|\bar{\mathcal{C}}_B|, \lceil (|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|)/2 \rceil\}$$

and

$$p(\overline{M}_H) + p(\bar{\mathcal{C}}) \leq |\overline{M}_u| + |\overline{M}_H|.$$

Proof. First consider the case that $\bar{\mathcal{C}} \setminus \mathcal{C}_B \neq \emptyset$. We know that such a required c' exists. This is given by Lemma 22 and Lemma 21 for classes in $\bar{\mathcal{C}}_{(1/2,3/4)} \setminus \mathcal{C}_B$ and $\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B$, respectively. The residual instance will be scheduled with the algorithm for instances without huge jobs. This generates a feasible schedule, since all machines that are non empty before the start of this subroutine have load at least 1. Furthermore only class c is partially scheduled and the load on m_0 can be rotated, such that $\hat{c}$ and $\check{c}$ do not overlap. This rotation is

always possible: The residual job c'' of the class is smaller than 3/4 and will therefore be scheduled consecutively by NoHuge. No matter when c'' gets scheduled, before or after it will be a large enough gap that fits c', since c' got scheduled starting at 0 (or ending at 3/2 after the rotation). Therefore, a correct rotation is possible.

In the case that $\bar{\mathcal{C}} \setminus \mathcal{C}_B = \emptyset$, we put the residual classes to individual machines. This is possible since only classes in $\mathcal{C}_B$ are left and the number of residual machines is at least $|\bar{\mathcal{C}}_B|$. □

Step 6. While $|\overline{M}_H| \geq 1$, $|\bar{\mathcal{C}}_{(1/2,3/4)} \cap \mathcal{C}_B| \geq 1$, and $|\bar{\mathcal{C}}_{\geq 3/4}| \geq 1$: Take $m_1 \in \overline{M}_H$, $b \in \bar{\mathcal{C}}_{(1/2,3/4)} \cap \mathcal{C}_B$ and $c \in \bar{\mathcal{C}}_{\geq 3/4}$. Open one new machine m_2. Schedule $\check{c}$ on m_1 such that it ends at 3/2. Schedule $\hat{c}$ on m_2 such that it starts at 0 and ends before 3/4. Schedule b at m_2 such that it ends at 3/2, see Fig. 4.4b. Close both machines. If $|\overline{M}_H| = 0$, continue with NoHuge on the residual instance.

Claim. *After this step all jobs are scheduled feasibly or* $|\overline{M}_H| \geq 1$ *and we have* $|\bar{\mathcal{C}} \setminus (\bar{\mathcal{C}}_{(1/2,3/4)} \cap \mathcal{C}_B)| = 0$ *or* $|\bar{\mathcal{C}} \setminus \bar{\mathcal{C}}_{\geq 3/4}| = 0$. *Furthermore, all jobs are scheduled feasibly in this step, all scheduled jobs are finished by* 3/2 *and for the residual empty machines* $\overline{M}_u$ *it holds that*

$$|\overline{M}_u| \geq \max\{|\bar{\mathcal{C}}_B|, \lceil (|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|)/2 \rceil\}$$

and

$$p(\overline{M}_H) + p(\bar{\mathcal{C}}) \leq |\overline{M}_u| + |\overline{M}_H|.$$

Proof. Note that we open one more machine in each iteration of the step. This machine has to exist, since in each of these steps, we have $|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B| \geq 2$. In this step, we have reduced $|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|$ by 2 and $|\bar{\mathcal{C}}_B|$ at least by 1. Hence there still have to exist $\max\{|\bar{\mathcal{C}}_B|, \lceil (|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|)/2 \rceil\}$ unused machines. In each iteration of this step, we close two machines but also reduce the residual load by at least $3/4 + 1/2 + 3/4 = 2$, proving the upper bound on the residual load.

Hence, if we have used NoHuge on the residual instance, by Lemma 23 it generates a schedule with makespan at most 3/2 because $p(\bar{\mathcal{C}}) \leq |\overline{M}_u|$ at that point and no class was scheduled partially. □

Step 7. If $|\bar{\mathcal{C}}_{(1/2,3/4)} \cap \mathcal{C}_B| \neq 0$, open one machine for each of these classes.

Claim. *After this step all jobs are feasibly scheduled or it holds that* $|\overline{M}_H| \geq 1$ *and all residual classes have a total processing time of at least* 3/4, *all scheduled jobs are finished by* 3/2 *and for the residual empty machines* $\overline{M}_u$ *it holds that*

$$|\overline{M}_u| \geq \max\{|\bar{\mathcal{C}}_B|, \lceil (|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|)/2 \rceil\}$$

and

$$p(\overline{M}_H) + p(\bar{\mathcal{C}}) \leq |\overline{M}_u| + |\overline{M}_H|.$$

Proof. Note that if $|\bar{\mathcal{C}}_{(1/2,3/4)} \cap \mathcal{C}_B| \neq 0$ this set is the only set containing unscheduled classes. Since we still have $|\overline{M}_u| \geq |\bar{\mathcal{C}}_B|$ unused machines, we can feasibly open one machine for each of these classes and are done. □

Step 8. While $|\overline{M}_H| \geq 2$ and $|\bar{\mathcal{C}}_{\geq 3/4}| \geq 2$: Take $m_1, m_2 \in \overline{M}_H$, $c_1, c_2 \in \bar{\mathcal{C}}_{\geq 3/4}$ starting with the classes in $\bar{\mathcal{C}}_B$. Shift all jobs on m_2 to the top, such that the last job ends at 3/2. Schedule $\check{c}_1$ on m_1 as one block that ends at 3/2 and all the jobs from $\check{c}_2$ as one block on m_2 that starts at 0. Open one more machine m_3 where we start the jobs from $\hat{c}_1$ at 0 and let the last job from $\hat{c}_2$ end at 3/2, see Fig. 4.4c. Close all three machines m_1, m_2, m_3. If $|\overline{M}_H| = 0$, continue with NoHuge on the residual instance.

Claim. *After this step all jobs are scheduled or it holds that either* $|\overline{M}_H| = 1$ *or* $|\bar{\mathcal{C}}_{\geq 3/4}| \leq 1$. *Furthermore* $|\bar{\mathcal{C}} \setminus \bar{\mathcal{C}}_{\geq 3/4}| = 0$ *and in each iteration the partial schedule is feasible, all scheduled jobs are finished by* 3/2 *and for the residual empty machines* $\overline{M}_u$ *it holds that*

$$|\overline{M}_u| \geq \max\{|\bar{\mathcal{C}}_B|, \lceil (|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|)/2 \rceil\}$$

and

$$p(\overline{M}_H) + p(\bar{\mathcal{C}}) \leq |\overline{M}_u| + |\overline{M}_H|.$$

Proof. In each of these steps, no two jobs from the same class overlap. Note that we open one more machine in each iteration of the step. This machine has to exist, since $|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B| \geq 2$ before this step. Since all remaining classes have load at least 3/4 it holds that $|\bar{\mathcal{C}}_B| = |\bar{\mathcal{C}}_B| \cap \bar{\mathcal{C}}_{\geq 3/4}$. Therefore, if $|\bar{\mathcal{C}}_B| \neq 0$ we used at least one such class and reduced $|\bar{\mathcal{C}}_B|$ by at least 1. We also reduced $|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|$ by 2 and hence there are still $\max\{|\bar{\mathcal{C}}_B|, \lceil (|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|)/2 \rceil\}$ unused machines. Lastly, in each of these steps, we close three machines but also reduce the residual load by at least 3 (4 classes with processing time at least 3/4 each), proving the upper bound on the residual load.

Hence, if we have used NoHuge on the residual instance, by Lemma 23 it generates a schedule with makespan at most 3/2 because $p(\bar{\mathcal{C}}) \leq |\overline{M}_u|$ at that point and no class was scheduled partially. □

Step 9. If $|\overline{M}_H| \geq 2$ or $|\bar{\mathcal{C}} \setminus \mathcal{C}_B| = 0$, open one machine for each of the remaining classes.

Claim. *After this step either all jobs are scheduled or it holds that* $|\overline{M}_H| = 1$, $|\bar{\mathcal{C}} \setminus \bar{\mathcal{C}}_{\geq 3/4}| = 0$, $\bar{\mathcal{C}} \setminus \mathcal{C}_B \neq \emptyset$, *the partial schedule is feasible, all scheduled jobs are finished by* 3/2, *and for the residual empty machines* $\overline{M}_u$ *it holds that*

$$|\overline{M}_u| \geq \max\{|\bar{\mathcal{C}}_B|, \lceil (|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|)/2 \rceil\}$$

and

$$p(\overline{M}_H) + p(\bar{\mathcal{C}}) \leq |\overline{M}_u| + |\overline{M}_H|.$$

Proof. Due to the previous steps, $|\overline{M}_H| \geq 2$ implies that $|\bar{\mathcal{C}}_{\geq 3/4}| \leq 1$, and if $|\bar{\mathcal{C}}_{\geq 3/4}| = 0$ we have already scheduled all the jobs. Otherwise, if it is true that $|\bar{\mathcal{C}}_{\geq 3/4}| = 1$, there has to be at least one unused machine because there are at least $\max\{|\bar{\mathcal{C}}_B|, \lceil (|\bar{\mathcal{C}}_B| + |\bar{\mathcal{C}}_{\geq 3/4} \setminus \mathcal{C}_B|)/2 \rceil\}$ unused machines.

If on the other hand $|\bar{\mathcal{C}} \setminus \mathcal{C}_B| = 0$, we still have $|\overline{M}_u| \geq |\bar{\mathcal{C}}_B|$ unused machines, we can feasibly open one machine for each of these classes and are done. □

Step 10. If $|\overline{M}_H| = 1$, take $c \in \bar{\mathcal{C}} \setminus \mathcal{C}_B$. It holds that $p(c) \geq 3/4$ and there exists $c' \in \{\hat{c}, \check{c}\}$ with $p(c') \in (1/4, 1/2]$. Place c' on $m_0 \in \overline{M}_H$. Continue with NoHuge to schedule the residual jobs including the job $c'' \in c \setminus \{c'\}$. Rotate the load on m_0 such that c' does not overlap with c''.

Claim. *After this step all jobs are scheduled feasibly and all scheduled jobs are finished by* 3/2.

Proof. The algorithm for instances without huge jobs, can feasibly finish the schedule with makespan at most 3/2 by Lemma 23 since $p(\bar{\mathcal{C}}) \leq |\overline{M}_u|$ at that point, all non empty machines have load at least 1 on average, and every class except c is either fully scheduled, or not scheduled at all. Like in Step 5, the rotation makes sure that there is no conflict within c. □

Lemma 24. *Given any instance $I = (m, \mathcal{C})$ of* MSRS*, ThreeHalf produces a feasible schedule with makespan at most $\frac{3}{2}\mathrm{OPT}(I)$.*

Proof. This is a direct consequence when considering the state after each step of the algorithm. □

4.4 APPROXIMATION SCHEMES

In this section, we present two approximation schemes for MSRS as follows.

Theorem 15. *There exists an* EPTAS *for* MSRS *if either the number m of machines is constant or $\lfloor \varepsilon m \rfloor$ additional machines may be used, i. e. some resource augmentation is allowed.*

To achieve these results, we follow a framework that was introduced in [67] and also used in [70]. In particular, we consider a simplified version of the problem and prove the existence of a certain well-structured solution with only bounded loss in the objective compared to an optimal solution. The problem of finding such a solution then can be formulated as an IP of a particular form. This IP can be solved efficiently using n-fold integer programming algorithms. Furthermore, we guarantee that the solution for the simplified problem can be used to derive a solution for the original one with only little loss in the objective value. The main challenge lies in the design of the well-structured solution and the proof of its existence. This also causes the limitations of our result: A certain group of jobs may cause problems in the respective construction and to deal with them we either use a more fine-grained approach, yielding a polynomial running time if m is constant, or place the respective jobs on (few) additional machines using resource augmentation.

4.4.1 *Simplification*

We use the standard technique of dual approximation (cf. Chapter 1), i. e. we apply a binary search to find a sufficient makespan guess T. The goal is then to either find a schedule of length $(1 + \mathcal{O}(\varepsilon))T$ or correctly report that no schedule of length T exists. We introduce parameters δ and μ and call jobs j *big*, *medium*, or *small*, if $p(j) \in (\delta T, T]$, $p(j) \in (\mu T, \delta T]$, or $p(j) \in (0, \mu T]$, respectively. Furthermore, we assume $\varepsilon < 1/2$.

CHOOSING THE PARAMETERS We set $\mu = \varepsilon^2\delta$ and choose δ depending on the instance and on whether we consider the case with a constant number of machines or not. If m is arbitrary, we choose $\delta \in \{\varepsilon, \varepsilon^2, \dots, \varepsilon^{2/\varepsilon^2}\}$ such that the following two conditions hold:

1. The total size of jobs j with size $p(j) \in (\mu T, \delta T]$ is at most $\varepsilon^2 mT$.
2. The total size of jobs j with size $p(j) \leq \delta T$ from classes in which these jobs have overall size in $(\mu T, \delta T]$ is at most $\varepsilon^2 mT$.

If m is fixed, we choose $\delta \in \{\varepsilon, \varepsilon^2, \dots, \varepsilon^{2m/\varepsilon}\}$ such that:

1. The total size of jobs j with size $p(j) \in (\mu T, \delta T]$ is at most εT.
2. The total size of jobs j with size $p(j) \leq \delta T$ from classes in which these jobs have overall size in $(\mu T, \delta T]$ is at most εT.

Such a choice is possible in both cases due to the pigeonhole principle.

REMOVING THE MEDIUM JOBS FOR FIXED m Let I be the input instance and I_1 the instance we get if we remove all the medium jobs.

Lemma 25. *Let m be a constant. If there is a schedule with makespan T' for I, then there is also a schedule with makespan T' for I_1; and if there is a schedule with makespan T' for I_1, then there is also a schedule with makespan $T' + \varepsilon T$ for I.*

Proof. The first implication is obvious. For the other direction, note that the overall size of the medium jobs is upper bounded by εT in this case and hence we can place all of them at the end of the schedule on some arbitrary machine. □

REMOVING THE MEDIUM JOBS FOR ARBITRARY m Let I be the input instance and I_1 the instance we get if we remove all the medium jobs from classes including at most εT medium load and the entire classes containing more than εT medium load.

Lemma 26. *Let m be part of the input. If there is a schedule with makespan T' for I, then there is also a schedule with makespan T' for I_1. Vice-versa, if there is a schedule with makespan T' for I_1, then there is also a schedule with makespan $T' + \varepsilon T$ for I using at most $\lfloor \varepsilon m \rfloor$ additional machines.*

Proof. The first direction is again obvious. For the other direction, we first consider the medium jobs from classes including at most εT medium load. We again place these jobs at the end of the schedule. In particular, they can be placed using the following greedy approach. We always place all the respective jobs belonging to the same class on the same machine and hence we may glue them together, i. e. assume that each class only contains one job (of size at most εT). The jobs are considered ordered decreasingly by size. On the current machine, we place the jobs starting at time T' one after another until the placement of the next job would result in a makespan greater than $T' + \varepsilon T$ or until no job is left. If there are jobs left, we continue on the next machine. Note that due to the ordering of the jobs, we can guarantee that each

machine on which we stopped placing jobs to avoid a makespan greater than $T' + \varepsilon T$, we can guarantee that they did receive a load of at least $\varepsilon T/2 > \varepsilon^2 T$ (using $\varepsilon < 1/2$). Since the overall load of the medium jobs is at most $\varepsilon^2 mT$, all the considered jobs can be placed.

Next, we consider the classes including more than εT medium load which were removed completely. Let $\mathcal{C}'$ be the set of classes containing more than εT medium load and let $p_m(c)$ denote the corresponding load of each $c \in \mathcal{C}'$. Then $|\mathcal{C}'|\varepsilon T < \sum_{c\in\mathcal{C}'} p_m(c) \leq \varepsilon^2 mT$ yielding $|\mathcal{C}'| < \varepsilon m$. Hence, we can place these classes on $\lfloor \varepsilon m \rfloor$ additional machines such that each machine receives exactly one class. Since we place the entire classes, this cannot cause conflicts. □

REMOVING SOME SMALL JOBS A (T', L')-schedule is a schedule with makespan at most T' and at least L' idle time throughout the schedule. Let I_2 be the instance we get if we remove all the small jobs from classes in which these jobs have overall size of at most δT from I_1. Let L be the overall size of the jobs removed in this step. We obviously have:

Lemma 27. *If there is a schedule with makespan T' for I_1, then there is also a (T', L)-schedule for I_2.*

LAYERED SCHEDULE AND ROUNDED PROCESSING TIMES Next, we will consider certain well-structured schedules called *layered schedules*. For some positive number ξ, we call a schedule ξ-layered, if the processing of each job starts at a multiple of ξ. The time between two such multiples is called a *layer* and the corresponding time on a single machine a *slot*.

In the following we show, that such a layered schedule can be generated, by rounding processing times and fusing jobs. Let I_3 be the instance we get if we round the processing times of the big jobs and replace the remaining small jobs with placeholders. In particular, let $p'(j) = \lceil p(j)/(\varepsilon\delta T)\rceil \varepsilon\delta T$ be the rounded size for each big job j. Furthermore, for each class $c \in \mathcal{C}$ with $p_s(c) = \sum_{j\in c, p(j)\leq \mu T} p(j) > \delta T$, we remove the small jobs and introduce $\lceil p_s(c)/(\varepsilon\delta T)\rceil$ new jobs with processing time $\varepsilon\delta T$ each.

Lemma 28. *If there is a (T', L)-schedule for I_2, then there is also an $\varepsilon\delta T$-layered $((1 + 2\varepsilon)T', L)$-schedule for I_3.*

Proof. Consider a (T', L)-schedule for I_2 and picture each job in a container. We stretch the schedule by a factor of $(1 + 2\varepsilon)$, and, in doing so, we move and stretch the containers correspondingly. The jobs inside the container, however, are not stretched. Note that each job can be moved inside its container without creating conflicts and we initially move each job to the bottom of its container. Next, we move each *big* job inside its container up such that it starts at the next layer border and increase its size to the rounded size. Since each big job j has size at least δT its container has size at least $p(j) + 2\varepsilon\delta T$ and therefore, each job remains inside its container. At this point, each big job starts and ends at a layer border and has the correct rounded size.

Next, we consider the small jobs. Let S be the set of small jobs in I_2 and n_c the number of placeholders we want to introduce for class c, i. e. $n_c = \lceil p_s(c)/(\varepsilon\delta T)\rceil$. For each class c, we grow the size of the small jobs inside their containers until the overall size of the jobs in $c \cap S$ is equal to $n_c\varepsilon\delta T$. This

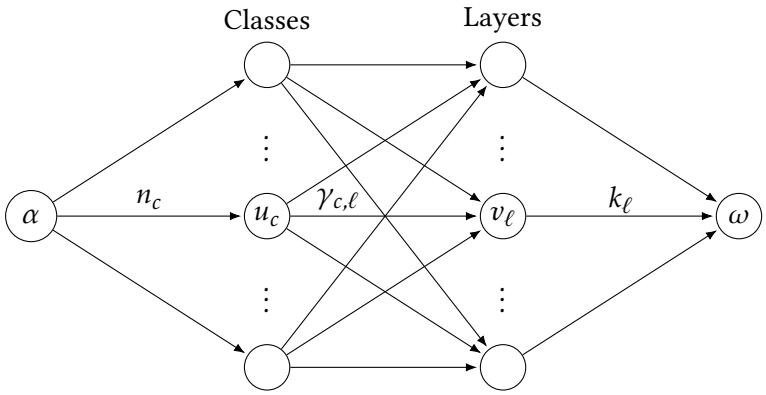


Figure 4.5: Flow network used in the construction of a layered schedule

is again possible since $p_s(c) > \delta T$. We denote the changed processing time of a small job j as p_j^*. Now, slots in which (parts of) some small job are placed can only contain small load due to the steps we performed for the big jobs. We will use the initial distribution of the small jobs as a starting point to find a feasible placement of the placeholder jobs in the layers. We add some notation. Let $\Xi = \{\ell \in \mathbb{Z}_{>0} \mid (\ell-1)\varepsilon\delta T \leq (1+2\varepsilon)T'\}$ be the set of layers, $\lambda(j,\ell)$ the fraction of job $j \in S$ placed in layer $\ell \in \Xi$ (i. e. $\sum_{\ell\in\Xi} \lambda_{j,\ell} = 1$), $p_\ell^* = \sum_{j\in S} \lambda_{j,\ell} p_j^*$ the small load placed in layer ℓ, $k_\ell = \lceil p_\ell^*/(\varepsilon\delta T)\rceil$ the rounded up number of slots needed for the overall small load in ℓ, and $\gamma_{c,\ell} = \lceil \sum_{j\in c\cap S} \lambda_{j,\ell} \rceil \in \{0,1\}$ a parameter indicating whether a small job belonging to class c is scheduled in layer ℓ. Note that we have $\sum_{\ell\in\Xi} \sum_{j\in S\cap c} \lambda(j,\ell) = n_c$. We now construct a flow network with integral capacities for which the given placement of the small jobs yields a maximum flow and utilize flow integrality to find a feasible placement for the placeholder. A very similar approach was taken in [67, 70]. The flow network is defined as follows and visualized in Fig. 4.5.

- There is a source α, a sink ω, a node u_c for each class c, and a node v_ℓ for each layer ℓ.
- The source is connected to each class node u_c via an edge (α, u_c) with capacity n_c.
- Each class node u_c is connected to each node v_ℓ via an edge (u_c, v_ℓ) with capacity $\gamma_{c,\ell}$.
- Each layer node v_ℓ is connected to the sink via an edge (v_ℓ, ω) with capacity k_ℓ.
- A maximum flow f is given by $f(\alpha, u_c) = n_c$, as well as $f(u_c, v_\ell) = \sum_{j\in S\cap c} \lambda(j,\ell)$, and $f(v_\ell, \omega) = \sum_{j\in S} \lambda(j,\ell)$.

It is easy to check that f is indeed a feasible maximum flow. Moreover, all capacities are integral and hence there exists an integral flow f' with the same value. We now remove all the small jobs from the schedule and assign the placeholders into slots according to f'. In particular, we assign a placeholder small job belonging to class c to a slot in layer ℓ if $f'(c,\ell) = 1$. Since slots that originally did receive some small jobs cannot contain any big load, the

definition of k_ℓ together with flow conservation imply that there are enough slots to do so. Furthermore, this cannot produce conflicts since each layer receives at most one placeholder job of each class and the presence of a big job of a class c in a layer ℓ implies $\gamma_{c,\ell} = 0$ and hence prevents the placement of a placeholder. Hence, we did construct a layered schedule with makespan at most $(1+2\varepsilon)T'$ for I_3. Finally note that the free space is preserved as well since jobs were only increased inside their containers after the stretching step. □

REINSERTING THE SMALL JOBS Finally, we discuss the reinsertion of all of the small jobs as well as the use of the original sizes.

Lemma 29. *If an $\varepsilon\delta T$-layered (T', L)-schedule for I_3, then there is also a schedule with makespan $(1+\varepsilon)T' + \varepsilon T$ for I_1.*

Proof. We first discuss the insertion of the small jobs starting with the ones from classes in which small jobs have overall size in $(\mu T, \delta T]$. Due to the choice of the medium jobs, the respective jobs have overall size of at most $\varepsilon^2 mT$ if m is part of the input or εT if m is constant. In the former, we can use a simple greedy procedure similar to the one in the proof of Lemma 26 to place them at the end of the schedule, and in the latter, we can just place all of them at the end of the schedule on an arbitrary machine. In either case, the objective value grows by at most εT.

Next, we stretch the schedule by a factor of $(1+\varepsilon)$ increasing the sizes of the jobs, layers, and free space accordingly. Now, each placeholder small job has a size of $(1+\varepsilon)\varepsilon\delta T = \varepsilon\delta T + \mu T$. Hence, if we remove a placeholder belonging to a class c and greedily place original small jobs of class c into the respective slot, we can guarantee that at least a load of $\varepsilon\delta T$ is placed (unless all of the small jobs of the class already have been placed). Finally, we place the jobs from classes in which the small jobs have an overall size of at most μT. If there is a big job in the same class, we fix one of them and decrease its size by μT (this is a smaller decrease than the increase due to the stretching step) and place the small jobs in the freed space. Else, we can place them greedily in the free slots placing all the jobs of the same class in the same slot (again utilizing that each free slot was increased by μT in the stretching step). This cannot produce conflicts since these classes do not contain any big jobs. In a last step we reduce the sizes of the big jobs to their original ones. □

4.4.2 *Integer Program*

To find a layered schedule, we utilize an IP approach. The corresponding IP is essentially a *module configuration IP* as introduced in [67] but, for the sake of simplicity, we diverge from the notation in the respective work. Considering Lemmas 25 to 28, we set $T' = (1+2\varepsilon)T$ and search for an $\varepsilon\delta T$-layered (T', L)-schedule for I_3. To this end we introduce some notation. Let $\Xi = \{\ell \in \mathbb{Z}_{>0} \mid (\ell-1)\varepsilon\delta T \leq (1+2\varepsilon)T'\}$ be the set of layers, P be the set of distinct processing times in I_3, and $n_p^{(c)}$ the number of jobs of size p in class c for each $p \in P$ and $c \in \mathcal{C}$. Furthermore, we define a (time) *window*

as a pair $(\ell, p) \in \Xi \times P$ of a starting layer ℓ and a processing time p, and a *configuration* K as a selection of windows $\{0,1\}^{\mathcal{W}}$ such that no two conflicting windows are chosen, i. e. $\sum_{(\ell,p)\in\mathcal{W}_{\ell'}} K_{\ell,p} \leq 1$ for each layer ℓ'. The set of configurations is denoted as $\mathcal{K}$, the set of windows as $\mathcal{W}$, and the the set of windows intersecting layer ℓ as $\mathcal{W}_\ell$. A window (ℓ, p) intersects $p/(\varepsilon\delta T)$ many succeeding layers starting with layer ℓ.

Observation 11. *We have*

$$|P| \in \mathcal{O}(1/(\varepsilon\delta)), |\Xi| \in \mathcal{O}(1/(\varepsilon\delta)), |\mathcal{W}| \in \mathcal{O}(1/(\varepsilon\delta)^2), \text{ and } |\mathcal{K}| \in 2^{\mathcal{O}(1/(\varepsilon\delta)^2)}.$$

Proof. Due to the rounding, the processing times are multiples of $\varepsilon\delta T$ and upper bounded by $T + \varepsilon\delta$. Hence, we have $|P| \in \mathcal{O}(1/(\varepsilon\delta))$ and the same argument yields $|\Xi| \in \mathcal{O}(1/(\varepsilon\delta))$ which directly implies $|\mathcal{W}| \in \mathcal{O}(1/(\varepsilon\delta)^2)$ which in turn yields $|\mathcal{K}| \in 2^{\mathcal{O}(1/(\varepsilon\delta)^2)}$. □

In the IP, we have a variable $x_K \in \{0, \dots, m\}$ for each $k \in \mathcal{K}$, a variable $y^{(c)}_{\ell,p} \in \{0, \dots, n\}$ for each class $c \in \mathcal{C}$ and window $(\ell, p) \in \mathcal{W}$, as well as the following constraints:

$$\sum_{K\in\mathcal{K}} x_K = m \tag{4.1}$$

$$\sum_{K\in\mathcal{K}} K_{\ell,p} x_K = \sum_{c\in\mathcal{C}} y^{(c)}_{\ell,p} \qquad \forall (\ell, p) \in \mathcal{W} \tag{4.2}$$

$$\sum_{\ell\in\Xi} y^{(c)}_{\ell,p} = n^{(c)}_p \qquad \forall c \in \mathcal{C}, p \in P \tag{4.3}$$

$$\sum_{(\ell',p)\in\mathcal{W}_\ell} y^{(c)}_{\ell',p} \leq 1 \qquad \forall c \in \mathcal{C}, \ell \in \Xi \tag{4.4}$$

The variables $y^{(c)}_{\ell,p}$ are used to reserve time windows for the placement of jobs belonging to class c, (4.3) guarantees that the correct number is chosen, and due to (4.4) placing the respective jobs in the windows will not create conflicts. Furthermore, the variables x_K are used to chose m configurations (due to (4.1)). Each such configuration corresponds to a scheduling pattern on one of the m machines. In particular, a configuration is by definition a selection of non-overlapping time windows and in (4.2) we make sure that these configurations cover the selected windows. Hence, it is easy to construct a solution for the IP given a $\varepsilon\delta T$-layered (T', L)-schedule and vice-versa yielding:

Lemma 30. *There exists an $\varepsilon\delta T$-layered (T', L)-schedule for I_3, if and only if the above IP is feasible.*

4.4.3 *Algorithm and Analysis*

Summing up, we use a binary search framework to get the makespan guess T, perform the simplification steps described in Lemmas 25 to 28, formulate and solve the described IP, construct a schedule from the IP solution, and transform it into a schedule for the original instance using the steps described

in Lemmas 25, 26, 28 and 29. For the given makespan guess T, we thus find a schedule with makespan at most $(1+\varepsilon)(1+2\varepsilon)T + 2\varepsilon T = (1+\mathcal{O}(\varepsilon))T$ or, if the IP is not feasible, correctly report that a schedule with makespan T does not exist.

Regarding the running time, it is easy to see that the critical step lies in solving the IP, since all the other ones mostly involve simple changes of the instance and fast greedy procedures that obviously run in polynomial time. Hence, we take a closer look at the IP and again essentially apply the approach introduced in [67], i. e. solving it via n-fold integer programming.

APPLYING n-FOLD INTEGER PROGRAMMING We have to slightly change our ILP in order to bring it into the form of an N-fold ILP (cf. Chapter 1). In particular, we copy the variables x_K such that each variable is present $|\mathcal{C}|$ many times but do not use any of these variables except for the original copy (hence the constraints are not changed). Furthermore, we introduce slack variables for Eq. (4.4) to transform the constraint into an equation. After performing these steps the number of blocks N is equal to $|\mathcal{C}|$, the number of block variables is given by $|\mathcal{K}| + |\mathcal{W}| + |\Xi|$, and the number of global and local constraints by $|\mathcal{W}| + 1$ (Eqs. (4.1) and (4.2)) and $|P| + |\Xi|$, respectively. Furthermore, the parameter Δ is equal to 1. Hence, by applying Theorem 1 the ILP can be solved in time $2^{\mathcal{O}(1/(\varepsilon\delta)^5 \log(1/(\varepsilon\delta)))}|\mathcal{C}|^{1+o(1)}$ doubly exponential in $1/\varepsilon$ where the former factor can be bounded by $2^{2^{\mathcal{O}(1/\varepsilon^2 \log(1/\varepsilon))}}$ and $2^{2^{\mathcal{O}(m/\varepsilon \log(1/\varepsilon))}}$ for arbitrary m and fixed m, respectively.

4.5 OPEN QUESTIONS

Of course, the most natural question is whether our approximation schemes may be refined to achieve a PTAS for MSRS for an arbitrary number of machines and without resource augmentation as well. Also, for the case with only a constant number of machines, an FPTAS is not ruled out at this point.

Moreover, it would be interesting to explore natural extensions of MSRS and, in particular, to investigate for which variants approximation schemes may or may not be possible. Note that MSRS can be seen as a special case of scheduling with conflicts where the conflict graph is a cograph. This problem is known to be NP-hard already for unit-sized jobs [22] and it would be interesting to explore inapproximability for arbitrary sizes. Regarding the design of approximation schemes, on the other hand, variants where the corresponding conflict graph is a particularly simple cograph may be interesting.

Part II

EXACT ALGORITHMS

5 RESPONSE TIME ANALYSIS AND MIXING SET

5.1 INTRODUCTION

We consider fixed-priority uniprocessor real-time task scheduling with constrained deadlines and task release jitters in the *sporadic task model* [99] which is a common model to analyze real-time task systems.

MODEL A sporadic task system $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ is a set of sporadic tasks τ_i which are given as a quadruple of non-negative integers $\tau_i = (c_i, d_i, p_i, \eta_i)$ where c_i represents the task's worst-case execution time, d_i its relative deadline, p_i its period, and η_i its release jitter. The task τ_i generates an infinite sequence of jobs. Each job has an execution time of at most $c_i \geq 1$ time units and a deadline exactly d_i time units after its arrival time, where $c_i \leq d_i \leq p_i$. The jobs of the task are considered to arrive separated in time by at least $p_i \geq 1$ time units. However, a job is not ready for execution until it is *released*. The time difference between the arrival time and the release time is called *release jitter*. Hence, in jitter-free systems the jobs are released in the moment they arrive, i. e. arrival time equals release time. The release jitter $\eta_i \geq 0$ of the task is the maximum time difference between the arrival times and the release times over all jobs of τ_i. Naturally, one can assume that $\eta_i < d_i \leq p_i$.

The task system $\mathcal{T}$ has *harmonic periods* iff $p_i \geq p_j$ implies that $p_i/p_j \in \mathbb{Z}$ for all tasks τ_i, τ_j. The *utilization* of a task τ_i is the quantity c_i/p_i and the utilization of the task set $\mathcal{T}$ is $\sum_{i \leq n} c_i/p_i$. The common assumption is that $\sum_{i \leq n} c_i/p_i \leq 1$ as otherwise there are job sequences of $\mathcal{T}$ which can not be scheduled by any algorithm. We refer to this utilization bound as the *schedulability utilization bound*. However, a necessary utilization bound is $\sum_{i<n} c_i/p_i < 1$ as otherwise there is no solution for worst-case response times at all. We refer to this bound as the *general utilization bound*.

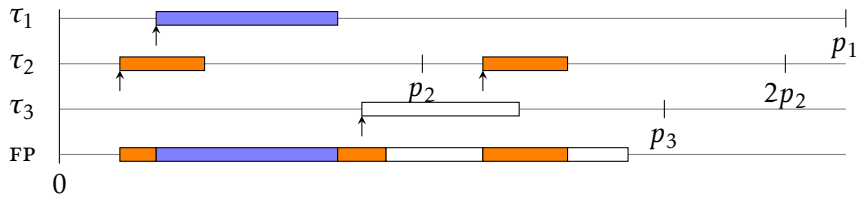


Figure 5.1: An example instance $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ with $\tau_1 = (15, d_1, 65, 8)$, $\tau_2 = (7, d_2, 30, 5)$, $\tau_3 = (13, d_3, 50, 25)$ and the resulting FP schedule. The arrows indicate the release times of the jobs.

FIXED-PRIORITY SCHEDULING In fixed-priority (FP) scheduling (also static-priority scheduling) the tasks are considered to have a predefined order of priorization. A task τ_i has a larger priority than a task τ_j if and only if $i < j$. An FP schedule preempts jobs according to these priorities; in more

detail, a running job is preempted on the release of a job of task of a higher priority to schedule the new job instead (see Fig. 5.1 for an example).

Two important special cases of FP scheduling are deadline monotonic (DM) scheduling [11, 8] and rate monotonic (RM) scheduling [48, 94, 111]. In DM scheduling earlier deadlines imply larger priorities while in RM scheduling smaller periods imply larger priorities. Both are equal if the deadline of each task is equal to its period.

RESPONSE TIME ANALYSIS AND SCHEDULABILITY TESTING The *response time* of a task is the maximum time difference between the release of a job and its completion. In a worst-case scenario we refer to them as *worst-case response times* and these may be used to decide the schedulability of a system. Since task release jitters intensify the complexity of worst-case response times and schedulability testing as well as other types of analysis, they are often assumed to be zero, i. e. $\eta_i = 0$ for all $i \leq n$. As pointed out by Liu and Layland [94], in such *jitter-free* systems the *critical instant* occurs when the jobs of all tasks τ_i are released simultaneously. See also synchronous arrival sequence (SAS) [13, 79]. Therefore, without release jitters the worst-case response time of task τ_j is the earliest point in time r_j where

$$r_j \geq c_j + \sum_{i=1}^{j-1} c_i \left\lceil \frac{r_j}{p_i} \right\rceil .$$

This inequality models the situation that all jobs are released at time 0. If $r_j \leq d_j$, then task t_j completes its job on time which means that task τ_j is schedulable. If $r_j \leq d_j$ for all tasks t_j, then the task system $\mathcal{T}$ is schedulable using the given priorities.

Here we consider systems with task release jitters. Audsley et al. [9] and Tindell et al. [115] have discussed the release jitter problem in detail. Their work leads to the following understanding of worst-case response times in systems with task release jitters. The worst-case response time of task τ_j is the earliest point in time r_j where

$$r_j \geq c_j + \sum_{i=1}^{j-1} c_i \left\lceil \frac{r_j + \eta_i}{p_i} \right\rceil .$$

The task is schedulable if $r_j \leq d_j - \eta_j$ and the task set $\mathcal{T}$ is schedulable if $r_j \leq d_j - \eta_j$ for all tasks τ_j. Without loss of generality we can assume that the execution requirement of each job of a task τ_i always equals the task's worst-case execution time c_i. Also we can assume that the i-th job of a task τ_j is released in the time interval $[(i-1)p_j, (i-1)p_j + \eta_j]$ and its due time is $(i-1)p_j + d_j$. Furthermore, we may restrict our interest to the response time r_n of the last task τ_n:

$$r_n = \min \{ t \mid t \geq c_n + \sum_{i<n} c_i \left\lceil \frac{t + \eta_i}{p_i} \right\rceil , t \in \mathbb{Z}_{\geq 0} \} . \tag{5.1}$$

We refer to the problem to compute r_n as *response time computation* (RTC).

For FP scheduling, RTC is NP-hard in general due to Eisenbrand and Rothvoß [45] even for $\eta = 0$ and it is even hard to approximate within a constant factor. However, for jitter-free task systems with harmonic periods it is polynomial which is a great result by Bonifaci et al. [24] and it may even be computed in near-linear time, cf. Nguyen et al. [102].

The computation of response times in real-time systems is often treated as a fixed point problem, where one aims for the smallest fixed point t to the fixed point equation $\Phi(t) = t$ where $\Phi(t) = c_n + \sum_{i<n} c_i \lceil (t + \eta_i)/p_i \rceil$. Especially, this allows to argue about intersections of Φ with the identity function. However, in this thesis we present a different approach.

MIXING SET In the MIXING SET problem one is given capacities $a \in \mathbb{Q}^n$ and a right-hand side $b \in \mathbb{Q}^n$ and the goal is to find a solution $(s, x) \in \mathbb{R}_{\geq 0} \times \mathbb{Z}^n$ which optimizes a linear objective function while satisfying the following system of inequalities:

$$s + a_i x_i \geq b_i \quad \forall i \in \{1, \ldots, n\}$$

The problem is NP-hard due to Eisenbrand and Rothvoß [46]. However, it can be solved in polynomial time in the case of unit capacities [57, 98] or harmonic capacities [118] (see also [30, 31] for simpler approaches). The MIXING SET problem plays an important role in production planning (in particular lot-sizing [108]).

RELATED WORK In the early '70s it was independently shown by Liu and Layland [94] and Serlin [111], that a set of n implicit-deadline synchronous periodic tasks is always schedulable in RM scheduling if the task's total utilization is bounded by $n(2^{1/n} - 1)$ which is $\ln(2) \approx 0.69$ for $n \to \infty$. Recently, this seminal result was complemented by Ekberg [48] who proved that $\ln(2)$ is in fact the boundary between polynomial and NP-hard schedulability testing; the FP schedulability problem is NP-hard even if it is restricted to task sets with RM priorities and utilization-bounded from above by any constant truly larger than $\ln(2)$.

Baruah [12] introduced the idea of ILP-tractability of schedulability analysis problems and in fact, our results prove tractability in his sense. As mentioned earlier, Eisenbrand and Rothvoß proved hardness results for both RTC [45] and the MIXING SET problem [46]; in fact, they proved reductions from directed diophantine approximation *to* these problems but they did not give any reductions from them or *between* them, which is what we have found.

RTC	p_i harmonic	$p_i \geq 1$
$\eta_i \leq p_i$	$\mathcal{O}(n^2(q + \log p_{\max}))$	$\mathcal{O}(n \operatorname{lcm}_{i<n} p_i)$ $\mathcal{O}(\max(Sn, T_{\mathcal{M}} \log p_{\max}))$
$\eta_i = 0$	$\mathcal{O}(n \log(n+p_{\max}))$ [24], $\mathcal{O}(n \log n)$ [102]	$\mathcal{O}(T_{\mathcal{M}} \log p_{\max})$
MIXING SET	a_i harmonic	$a_i \geq 1$
$b_i \in \mathbb{Z}$	$\mathcal{O}(n^2)$ [31]	$\mathcal{O}(n \operatorname{lcm}_{i \leq n} a_i)$ $\mathcal{O}(T_{\mathcal{R}} \log b_{\max})$
$b_i = \beta \geq \operatorname{lcm}_{j \leq n} a_j$	$\mathcal{O}(n \log(n + \beta))$	$\mathcal{O}(T_{\mathcal{R}} \log \beta)$

Table 5.1: Overview of algorithmic results for RTC and MIXING SET. Here $S \leq (\sum_{i<n} c_i)/(1-U)$ for utilization $U = \sum_{i<n} c_i/p_i$ is an upper bound on the value of s in any optimal solution to an instance of MIXING SET, $T_{\mathcal{M}}$ and $T_{\mathcal{R}}$ are upper bounds on the time required to solve an instance of MIXING SET or RTC, respectively, and $q \leq |\{ p_i - \eta_i \mid i \leq n \}| \leq n$. All results without a reference are given in this chapter

OUR CONTRIBUTION Here we give a short summary about all of our results.

(a) We present a simple dualization technique which allows to solve inequality-constrained optimization problems in a binary search by consecutively solving a dual formulation of the associated decision problem (see Section 5.2.1).

(b) **Response Times**

1. We establish new bounds for worst-case response times in FP real-time task systems with period-constrained task release jitter (see Section 5.2.3).
2. We present a conditional Karp reduction from RTC to the MIXING SET problem (see Section 5.3.1). We show how it can be used to derive a bunch of algorithmic results for both problems (see Table 5.1 for an overview).
3. For the most important case of harmonic periods we prove a Cook reduction from RTC to MIXING SET (see Section 5.3.3).
4. Usually, algorithmic results for worst-case response times are achieved for special cases like equal or harmonic periods only; in fact, we can cope even with the general case of *arbitrary* periods. Especially, we give a Turing reduction from RTC to MIXING SET and we study the dependence of the running times on the utilization (see Section 5.3.4).

(c) **Mixing Set** We can reverse our methodology to solve the MIXING SET problem by computing worst-case response times for associated real-time task systems (see Section 5.4).

(d) **4-block Programs** Finally, we show how the dualization technique can by applied to solve specific 4-block integer programs (see Section 5.6). Furthermore, we show that they are NP-hard to approximate to any constant factor (see Section 5.5).

RTC WITH HARMONIC PERIODS Here we want to highlight our result for harmonic periods, as it gives an efficient algorithm to a well-studied case of a long-standing problem. On the one hand, the release jitter problem was studied for about 30 years now (e. g. [9, 115, 37]), and also harmonic periods have been studied (e. g. [24, 102]) but on the other hand and to the best of our knowledge, there was no polynomial-time algorithm known before. Also remark that FP scheduling superceeds the generality of DM and RM scheduling and in fact, we never rely on their properties. In the following we give a brief overview about our algorithm for harmonic periods.

We consider the decision problem $r_n \leq_? k$ for which we present a Karp reduction to the MIXING SET problem in constant time. However, this reduction works for large values of k only, i. e. $k \geq S$ for some value S. In the case of harmonic periods we managed to develop an algorithm which discovers variable values which allow to reduce S such that smaller values for k can be decided. To this end we prove a structural property, which, given only a rough knowledge about r_n, reveals the exact variable values of the variables for the tasks of largest period. The algorithm then works in two steps. First, we do a walk over the differences $p_i - \eta_i$ in non-decreasing order to identify two consecutive differences $k_i = p_i - \eta_i$ and $k_{i+1} = p_{i+1} - \eta_{i+1}$ such that $r_n \in (k_i, k_{i+1}]$. Here, the order is of utmost importance, as we rely on the information of the previous decisions. In the second step we utilize a binary search to compute r_n. Together, this yields a Cook reduction to MIXING SET.

NOTATION Given a vector $v = (v_1, \ldots, v_n) \in \mathbb{R}^n$ we denote the largest or smallest entry in v by $v_{\max} = \max_{i \leq n} v_i$ or $v_{\min} = \min_{i \leq n} v_i$, respectively. For sizes a, b we write $a \leq_? b$ to denote the decision problem to decide whether $a \leq b$ is true or not, i. e. the correct answer to $a \leq_? b$ is YES if $a \leq b$ and NO otherwise. In the same manner we define $a \geq_? b$. For any positive integers $z_i \in \mathbb{Z}_{\geq 1}$ and an index set I let $\operatorname{lcm}_{i \in I} z_i$ denote the least common multiple of all numbers in $\{ z_i \mid i \in I \}$.

5.2 PRELIMINARIES

Here we introduce a dualization technique for optimization problems. Also we show first insights about the MIXING SET problem and present some new bounds for response times in real-time systems.

5.2.1 *A Dualization Technique*

We present a simple optimization technique by studying a dual formulation of the associated decision problem. Consider a general optimization problem as follows. Let X be a set and let $f : X \to \mathbb{Z}$ and $g : X \to \mathbb{R}$ be two computable functions and we seek to minimize $f(x)$ over all $x \in X$ while $g(x) \geq b$ for

some right-hand side $b \in \mathbb{R}$ and other constraints $G(x)$ have to be satisfied, i. e. we aim to compute

$$\sigma = \min\{f(x) \mid g(x) \geq b,\ G(x),\ x \in X\}.$$

For an integer k it holds that $\sigma \leq k$ if and only if

$$\max\{g(x) \mid f(x) \leq k,\ G(x),\ x \in X\} \geq b. \tag{5.2}$$

Hence, if we can bound σ to an interval $[\ell, u]$ and have an algorithm to decide (5.2) in at most time T for each $k \in [\ell, u]$, then we can compute σ in time $\mathcal{O}(T \log(u-\ell))$ by using a binary search for the smallest feasible k in $[\ell, u]$. At first glance this does not help a lot, as we have replaced one optimization problem by another. However, if f is a function of low complexity, then in order to decide (5.2) we can try to utilize an algorithm for $\max\{g(x) \mid G(x), x \in X\}$ with respect to the simple additional constraint $f(x) \leq \kappa$ (which is of low complexity). Remark that the approach may be adapted to approximate σ for a function $f : X \to \mathbb{R}$ as well. However, in this chapter we only care about integer-valued objectives. The technique is applied in Sections 5.3 and 5.6.

5.2.2 *Mixing Set*

Although Mixing Set can be solved in more general settings, for the scope of this chapter we will always assume that $a \in \mathbb{Z}^n_{\geq 1}$ and $b \in \mathbb{Z}^n$. In more detail, given capacities $a \in \mathbb{Z}^n_{\geq 1}$, a right-hand side $b \in \mathbb{Z}^n$, and weights $w \in \mathbb{Z}^n_{\geq 0}$ (and $w_0 \in \mathbb{Z}_{\geq 0}$) we consider the following minimization problem:

$$\min\{w_0 s + \textstyle\sum_{i \leq n} w_i x_i \mid s + a_i x_i \geq b_i\ \forall i \in [n],\quad s \in \mathbb{Z}_{\geq 0},\quad x \in \mathbb{Z}^n\} \tag{Mix}$$

Remark that this is a special case of a 2-stage stochastic ILP (see Chapter 1 and Theorem 2) with inequality constraints and block size 1×1 where all entries of the first column of the constraint matrix are 1 and the capacities a_i are placed on the diagonal.

Formally, this is not a Mixing Set problem, since $s \in \mathbb{Z}_{\geq 0}$ is required to be an integer instead of a real number. However, due to the integrality of all other numbers the relaxation to $s \in \mathbb{R}_{\geq 0}$ admits just the same set of *optimal* solutions. From [118, 30, 31] we can derive the following theorem.

Theorem 16 ([118, 30, 31]). *For harmonic capacities,* (Mix) *can be computed in time* $\mathcal{O}(n^2)$.

Given a value for s it is obvious that $x_i = \lceil (b_i - s)/a_i \rceil$ for all $i \leq n$ leads to an optimal completion of s to a solution (s, x) to (Mix). The solution (s, x) might not be optimal overall but the choice of x is optimal with respect to s. Therefore, given a solution (s, x) we will assume that $x_i = \lceil (b_i - s)/a_i \rceil$. Furthermore, we we will write $x_i(s) = \lceil (b_i - s)/a_i \rceil$ (and $x(s) = (x_1(s), \ldots, x_n(s))$) to denote the dependency of variable x_i on s. Remark that by adding/sub-

tracting the least common multiple $m = \operatorname{lcm}_{j\le n} a_j$ to/from s the value $x_i(s)$ is shifted by m/a_i in the sense that

$$x_i(s \pm m) = \left\lceil \frac{b_i - s \mp m}{a_i} \right\rceil = \left\lceil \frac{b_i - s}{a_i} \right\rceil \mp \frac{m}{a_i} = x_i(s) \mp \frac{m}{a_i} \quad \text{for all } i \le n.$$

We will depend on an upper bound on the value of s in optimal solutions to (Mix). Therefore, the following observations will be important.

Observation 12. *If $\sum_{i\le n} \frac{w_i}{a_i} > w_0$, then* (Mix) *is unbounded.* □

Proof. Let $m = \operatorname{lcm}_{i\le n} a_i$. For any feasible solution (s, x) we get another feasible solution by adding a multiple of m to s. In particular it holds that

$$\begin{aligned} & w_0(s+m) + \sum_{i\le n} w_i x_i(s+m) \\ &= w_0(s+m) + \sum_{i\le n} w_i x_i(s) - \sum_{i\le n} w_i \frac{m}{a_i} \\ &= w_0 s + \sum_{i\le n} w_i x_i(s) + m\left(w_0 - \sum_{i\le n} \frac{w_i}{a_i}\right) \end{aligned}$$

and that means that solution $(s', x(s'))$ with $s' = s + m > s$ will be a truly better solution than (s, x) if $\sum_{i\le n} \frac{w_i}{a_i} > w_0$. □

Hence, we assume that $\sum_{i\le n} \frac{w_i}{a_i} \le w_0$ in general.

Observation 13. *There is an optimal solution to* (Mix) *with $s < \operatorname{lcm}_{i\le n} a_i$.*

Proof. Let $m = \operatorname{lcm}_{i\le n} a_i$ and let (s, x) be an optimal solution with $s \ge m$. We find

$$\begin{aligned} & w_0(s-m) + \sum_{i\le n} w_i x_i(s-m) \\ &= w_0 s + \sum_{i\le n} w_i x_i(s) - m\underbrace{\left(w_0 - \sum_{i\le n} \frac{w_i}{a_i}\right)}_{\ge 0} \\ &\le w_0 s + \sum_{i\le n} w_i x_i(s). \end{aligned}$$

Hence, $(s' = s - m, x' = x(s'))$ is an optimal solution too. Iterate this argument to find an optimal solution with s smaller than m. □

By using an analog contradiction proof it is easy to see that *any* optimal solution will hold $s < \operatorname{lcm}_{i\le n} a_i$ if $\sum_{i\le n} \frac{w_i}{a_i}$ is truly smaller than w_0. We will use this strengthening:

Observation 14. *If $\sum_{i\le n} \frac{w_i}{a_i} < w_0$, then any optimal solution to* (Mix) *holds $s < \operatorname{lcm}_{i\le n} a_i$.*

In Section 5.3.2 we show that this bound on s is tight in general.

5.2.3 *Response time bounds*

First of all, we refine the utilization bound $\sum_{i<n} c_i/p_i < 1$. In general it holds that

$$\sum_{i<n} \frac{c_i}{p_i} \leq 1 - \frac{1}{\operatorname{lcm}_{i<n} p_i}. \tag{5.3}$$

To see this, let $m = \operatorname{lcm}_{j<n} p_j$. From $\sum_{i<n} c_i/p_i < 1$ it follows that $m \cdot \sum_{i<n} c_i/p_i < m$ and both sides of this inequality are integers which implies that $m \cdot \sum_{i<n} c_i/p_i \leq m-1$ and division by m yields the claim.

In the following we give bounds to worst-case response times for both the presence and absence of task release jitters. We start with the former case.

Lemma 31. *It holds that* $\ell \leq r_n \leq u = \min\{u_1, u_2\}$ *where*

$$\ell = \frac{c_n + \sum_{i<n} \frac{\eta_i}{p_i} c_i}{1 - \sum_{i<n} \frac{c_i}{p_i}},$$

and

$$u_1 = \ell + \frac{\sum_{i<n} c_i}{1 - \sum_{i<n} \frac{c_i}{p_i}},\ u_2 = \left\lceil \frac{\sum_{i \leq n} c_i}{(1 - \sum_{i<n} \frac{c_i}{p_i}) \operatorname{lcm}_{i<n} p_i} \right\rceil \operatorname{lcm}_{i<n} p_i.$$

Proof. By dropping the roundings it is easy to see that

$$r_n \;=\; c_n + \sum_{i<n} c_i \left\lceil \frac{r_n + \eta_i}{p_i} \right\rceil \;\geq\; c_n + r_n \sum_{i<n} \frac{c_i}{p_i} + \sum_{i<n} \frac{\eta_i}{p_i} c_i$$

which directly implies the lower bound ℓ. To prove the former upper bound u_1 we see that

$$r_n \;=\; c_n + \sum_{i<n} \left\lceil \frac{r_n + \eta_i}{p_i} \right\rceil c_i \;\leq\; c_n + \sum_{i<n} \left(\frac{r_n + \eta_i}{p_i} + 1 \right) c_i$$

and rearrangement yields

$$r_n \;\leq\; \ell + \frac{\sum_{i<n} c_i}{1 - \sum_{i<n} \frac{c_i}{p_i}} \;=\; u_1.$$

To confirm the latter upper bound u_2 we use the simple fact that $u_2 \geq (\sum_{i \leq n} c_i)/(1 - \sum_{i<n} c_i/p_i)$ and thus $\sum_{i \leq n} c_i \leq u_2(1 - \sum_{i<n} c_i/p_i)$ and this leads to

$$
\begin{aligned}
& c_n + \sum_{i<n} \left\lceil \frac{u_2 + \eta_i}{p_i} \right\rceil c_i \\
&= c_n + \sum_{i<n} \left(\frac{u_2}{p_i} + \left\lceil \frac{\eta_i}{p_i} \right\rceil \right) c_i && \text{lcm property} \\
&\leq c_n + \sum_{i<n} \left(\frac{u_2}{p_i} + 1 \right) c_i \;=\; u_2 \sum_{i<n} \frac{c_i}{p_i} + \sum_{i \leq n} c_i && 0 \leq \eta_i < p_i \\
&\leq u_2 \sum_{i<n} \frac{c_i}{p_i} + u_2 \left(1 - \sum_{i<n} \frac{c_i}{p_i} \right) \;=\; u_2
\end{aligned}
$$

which proves the claim. □

We will use the bound u_1 to show that the length of the interval $[\ell, u]$ is pseudo-polynomial and bound u_2 to prove that u is tight. We start with the former goal.

Lemma 32. *By using the general utilization bound it follows that $u - \ell \leq p_{\max}^n$. With the schedulability utilization bound it follows that $u - \ell \leq p_{\max}^2$ and even $u \leq 2p_{\max}^2$.*

Proof. The general utilization bound (5.3), i. e. $\sum_{i<n} c_i/p_i \leq 1 - 1/\mathrm{lcm}_{i<n} p_i$, implies that $\sum_{i<n} c_i \leq p_{\max} \sum_{i<n} c_i/p_i < p_{\max}$ and

$$
u_1 - \ell = \frac{\sum_{i<n} c_i}{1 - \sum_{i<n} \frac{c_i}{p_i}} \leq (\mathrm{lcm}_{i<n} p_i) \sum_{i<n} c_i \leq (\mathrm{lcm}_{i<n} p_i) p_{\max} \leq p_{\max}^n.
$$

Assume that $\sum_{i \leq n} c_i/p_i \leq 1$ which implies that $1/(1 - \sum_{i<n} c_i/p_i) \leq p_n/c_n$. It holds that $\sum_{i \leq n} c_i \leq p_{\max} \sum_{i \leq n} c_i/p_i \leq p_{\max}$ which implies that

$$
u_1 - \ell = \frac{\sum_{i<n} c_i}{1 - \sum_{i<n} \frac{c_i}{p_i}} = \frac{p_n}{c_n} \sum_{i<n} c_i \leq \frac{p_n}{c_n} p_{\max} \leq p_{\max}^2
$$

and by also using the definition of ℓ it follows that

$$
u_1 = \frac{\sum_{i \leq n} c_i + \sum_{i<n} \frac{\eta_i}{p_i} c_i}{1 - \sum_{i<n} \frac{c_i}{p_i}} \leq \frac{p_n}{c_n} \left(p_{\max} + \sum_{i<n} c_i \right) \leq \frac{p_n}{c_n} \cdot 2p_{\max} \leq 2p_{\max}^2
$$

and that proves the claim. □

Lemma 33. *The upper bound of Lemma 31 is tight, even for harmonic periods.*

To give the proof to Lemma 33 we will first prove the following constructive Lemma.

Lemma 34. *Any combination of $n-2$ integer processing times $c_1, \ldots, c_{n-2} \geq 1$, an integer period $p_1 > c_1$, and release jitters $\eta_1, \ldots, \eta_n$ can be supplemented to a sporadic task system of n tasks with full utilization and harmonic periods such that $c_n = 1$ and $p_{n-1} = p_n = p_{\max}$.*

Proof. Let $c_1, \dots, c_{n-2} \geq 1$ and $p_1 > c_1$ be given. We supplement the task system as follows. Set $p_i = (c_i+1)\cdot p_{i-1}$ for all $i \in \{2, \dots, n-2\}$, $p_{n-1} = 2p_{n-2}$, and $p_n = p_{n-1}$, as well as $c_{n-1} = 2p_{n-2}(1 - \sum_{i=1}^{n-2} \frac{c_i}{p_i}) - 1$ and $c_n = 1$.

Apparently, the periods are harmonic and $p_n = p_{\max}$. Thus, especially c_{n-1} is integral, since $c_{n-1} = 2(p_{n-2} - \sum_{i=1}^{n-2} \frac{p_{n-2}}{p_i} c_i) - 1$ and p_{n-2}/p_i is an integer for all $i \in [n-2]$. Clearly, we have $1 \leq c_i \leq p_i$ for all $i \in [n-2]$ and it holds that $c_{n-1} < 2p_{n-2} = p_{n-1}$. To see that $c_{n-1} \geq 1$, we show that

$$p_j\Big(1 - \sum_{i=1}^{j} \frac{c_i}{p_i}\Big) \geq 1 \qquad \text{for all } j \in [n-2]. \tag{5.4}$$

For $j = 1$ this means that $p_1(1 - c_1/p_1) \geq 1$ which is trivially satisfied by the choice of p_1. If the inequality holds for some task $j \in [n-3]$, then it holds that

$$p_{j+1}\Big(1 - \sum_{i=1}^{j+1} \frac{c_i}{p_i}\Big) = (c_{j+1}+1) \underbrace{p_j\Big(1 - \sum_{i=1}^{j} \frac{c_i}{p_i}\Big)}_{\geq 1} - c_{j+1} \geq 1,$$

so the inequality holds for task $j+1$ as well, and that ends the proof of (5.4). Hence, we get $c_{n-1} \geq 2 \cdot 1 - 1 = 1$. The defined task system has full utilization, i. e. $\sum_{i \leq n} \frac{c_i}{p_i} = 1$, since

$$\begin{aligned}
\sum_{i \leq n} \frac{c_i}{p_i} &= \sum_{i=1}^{n-2} \frac{c_i}{p_i} + \frac{c_{n-1}}{p_{n-1}} + \frac{c_n}{p_n} \\
&= \sum_{i=1}^{n-2} \frac{c_i}{p_i} + \underbrace{2\frac{p_{n-2}}{p_{n-1}}}_{=1}\Big(1 - \sum_{i=1}^{n-2} \frac{c_i}{p_i}\Big) - \frac{1}{p_{n-1}} + \frac{c_n}{p_n} \\
&= 1 - \frac{c_n}{p_n} + \frac{c_n}{p_n} = 1
\end{aligned}$$

where we used the definition of c_{n-1} as well as $p_{n-1} = 2p_{n-2}$, $c_n = 1$, and $p_n = p_{n-1}$. □

Proof of Lemma 33. We set $\eta_i = p_i$ for all $i \leq n$. According to Lemma 34 we can always construct an instance with these release jitters, harmonic periods, and full utilization, i. e. $\sum_{i \leq n} \frac{c_i}{p_i} = 1$, and also $c_n = 1$ and $p_{n-1} = p_n = p_{\max}$. Turning to Lemma 31 we see that such an instance holds

$$\ell = \frac{c_n + \sum_{i<n} \frac{\eta_i}{p_i} c_i}{1 - \sum_{i<n} \frac{c_i}{p_i}} = \frac{c_n + \sum_{i<n} \frac{p_i}{p_i} c_i}{1 - \sum_{i<n} \frac{c_i}{p_i}} = \frac{\sum_{i \leq n} c_i}{1 - \sum_{i<n} \frac{c_i}{p_i}} = u_2$$

where the last equality follows as ℓ is an integer multiple of $\operatorname{lcm}_{i<n} p_i$ since

$$\frac{1}{1 - \sum_{i<n} \frac{c_i}{p_i}} = \frac{1}{\frac{c_n}{p_n}} = \frac{p_n}{c_n} = p_n = p_{n-1} = \max_{i<n} p_i = \operatorname{lcm}_{i<n} p_i$$

and that completes the proof. □

We want to highlight that the instances of Lemma 33 (or Lemma 34 respectively) are also extreme for the case without release jitters, i. e. $\eta = 0$. Then the tight upper bound is $P = \operatorname{lcm}_{i \le n} p_i$; in fact, the upper bound of $c_n \cdot P$, which was given by Bonifaci et al. [24, Lemma 1], can be improved to P. The proof is short:

Lemma 35. *If $\eta = 0$, then $c_n/(1 - \sum_{i<n} \frac{c_i}{p_i}) \le r_n \le P$.*

Proof. The lower bound is implied by Lemma 31. For the upper bound see that

$$\begin{aligned} c_n + \sum_{i<n} \left\lceil \frac{P}{p_i} \right\rceil c_i &= c_n + P \cdot \sum_{i<n} \frac{c_i}{p_i} \\ &= p_n \cdot \frac{c_n}{p_n} + P \cdot \sum_{i<n} \frac{c_i}{p_i} \le P \cdot \sum_{i=1}^{n} \frac{c_i}{p_i} \le P. \end{aligned}$$

□

5.3 RESPONSE TIME COMPUTATION

We introduce a generalized problem formulation as follows. For a selection of tasks $I \subseteq [n-1]$ and an integer $\gamma \ge 1$ we aim to compute

$$\mathcal{R}(I, \gamma) = \min \{ t \mid t \ge \gamma + \textstyle\sum_{i \in I} c_i \left\lceil \frac{t + \eta_i}{p_i} \right\rceil,\ t \in \mathbb{Z}_{\ge 0} \}. \tag{5.5}$$

Obviously, $r_n = \mathcal{R}([n-1], c_n)$ is the worst-case response time of task τ_n.

5.3.1 *Conditional Karp reduction*

In the following we apply the dualization technique of Section 5.2.1 to RTC. We write $x \in \mathbb{Z}^I$ to introduce a variable $x_i \in \mathbb{Z}$ for each $i \in I$. Apparently, (5.5) may also be stated using the following formulation as an integer linear program:

$$\begin{aligned} &\mathcal{R}(I, \gamma) \\ &= \min \{ t \mid t \ge \gamma + \sum_{i \in I} c_i x_i,\ p_i x_i \ge t + \eta_i \ \forall i \in I,\ t \in \mathbb{Z}_{\ge 0},\ x \in \mathbb{Z}^I \} \quad (\mathcal{R}) \end{aligned}$$

Let u be the upper bound of Lemma 31 and let $k \in [u]$. One can readily verify that the associated decision problem $\mathcal{R}(I, \gamma) \le_? k$ may be decided by answering $\overline{\mathcal{R}}(I, k) \ge_? \gamma$ where

$$\begin{aligned} &\overline{\mathcal{R}}(I, k) \\ &= \max \{ t - \sum_{i \in I} c_i x_i \mid t \le k, p_i x_i \ge t + \eta_i \ \forall i \in I,\ t \in \mathbb{Z}_{\ge 0},\ x \in \mathbb{Z}^I \}. \quad (\overline{\mathcal{R}}) \end{aligned}$$

Let S be the smallest upper bound on the value of s in any optimal solution to the following instance of Mixing Set:

$$\begin{aligned}&\mathcal{M}(I,k)\\&= \min\{s + \sum_{i\in I} c_i x_i \mid s + p_i x_i \geq k + \eta_i \ \forall i \in I,\ s \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^I\}\end{aligned} \tag{$\mathcal{M}$}$$

We will prove that, if $k \geq S$ holds true, then $\overline{\mathcal{R}}(I,k) = k - \mathcal{M}(I,k)$. Hence, if k is large enough, i. e. $k \geq S$, we can decide $\mathcal{R}(I,\gamma) \leq_? k$ by using the following equivalence:

$$\begin{aligned}\mathcal{R}(I,\gamma) \leq k \quad &\Leftrightarrow \quad \overline{\mathcal{R}}(I,k) \geq \gamma\\ &\Leftrightarrow \quad k - \mathcal{M}(I,k) \geq \gamma \quad \Leftrightarrow \quad \mathcal{M}(I,k) \leq k - \gamma\end{aligned} \tag{5.6}$$

Thus, we get the following theorem.

Theorem 17. *Let $I \subseteq [n-1]$ and $\gamma \in \mathbb{Z}_{\geq 1}$. If $k \geq S$, then there is a reduction from $\mathcal{R}(I,\gamma) \leq_? k$ to $\mathcal{M}(I,k) \leq_? k - \gamma$ in constant time.*

In the case of harmonic periods we get harmonic capacities in the instance of Mixing Set which implies an efficient decision algorithm. However, without further insights we can decide $\mathcal{R}(I,\gamma) \leq_? k$ only for large values of k. In Section 5.3.3 we describe how to overcome this issue in the case of harmonic periods.

If $k \geq S$ holds, remark that optimal solutions to ($\mathcal{M}$) also do hold $x \in \mathbb{Z}^I_{\geq 0}$, since $s \leq S \leq k$ which implies $x_i = \lceil (k - s + \eta_i)/p_i \rceil \geq \lceil \eta_i/p_i \rceil \geq 0$ for all $i \in I$. By using Observation 14 it holds that $S \leq \operatorname{lcm}_{i\in I} p_i$, as we have that $\sum_{i\in I} \frac{c_i}{p_i} \leq \sum_{i<n} \frac{c_i}{p_i} < 1$.

Lemma 36. *If $k \geq S$, then $\overline{\mathcal{R}}(I,k) = k - \mathcal{M}(I,k)$.*

Proof. Let (t,x) be an optimal solution to ($\overline{\mathcal{R}}$) with objective value $k - \sigma$ for some value σ. We know that $0 \leq t \leq k$. Then by setting $s = k - t \geq 0$ we see that (s,x) is a solution to ($\mathcal{M}$) since $s + p_i x_i = k - t + p_i x_i \geq k - t + t + \eta_i = k + \eta_i$. Its objective value is

$$s + \sum_{i\in I} c_i x_i = k - t + \sum_{i\in I} c_i x_i = k - \left(t - \sum_{i\in I} c_i x_i\right) = k - (k - \sigma) = \sigma.$$

Vice-versa let (s,x) be an optimal solution to ($\mathcal{M}$) with objective value σ. We know that $0 \leq s \leq S \leq k$. Hence, by setting $t = k - s \geq 0$ we see that (t,x) is a solution to ($\overline{\mathcal{R}}$) since $t + \eta_i = k - s + \eta_i = k + \eta_i - s \leq s + p_i x_i - s = p_i x_i$ and $t = k - s \leq k$. Its objective value is

$$t - \sum_{i\in I} c_i x_i = k - s - \sum_{i\in I} c_i x_i = k - \left(s + \sum_{i\in I} c_i x_i\right) = k - \sigma$$

and that proves the claim. □

5.3.2 *The Bound on S is tight in general*

Here we prove that in general $\operatorname{lcm}_{i\le n} a_i - 1$ is indeed the best upper bound on S. Given an arbitrary number $n \in \mathbb{Z}_{\ge 2}$ we define an instance of Mixing Set by $w_i = 2^i$, $a_i = n \cdot 2^i$, and $b_i = n \cdot 2^n - 1$ for all $i \in [n]$. Remark that $\operatorname{lcm}_{i\le n} a_i = a_n = n \cdot 2^n$ and $\sum_{i\le n} w_i/a_i = \sum_{i\le n} 1/n = 1$. We get the objective function

$$f(s) = s + \sum_{i\le n} w_i \left\lceil \frac{b_i - s}{a_i} \right\rceil = s + \sum_{i\le n} 2^i \left\lceil \frac{n\cdot 2^n - 1 - s}{n\cdot 2^i} \right\rceil.$$

First consider $0 \le s \le n \cdot 2^n - 2$. We have $1 \le n \cdot 2^n - 1 - s \le n \cdot 2^n - 1$ and at least for $i = n$ this yields

$$\left\lceil \frac{n\cdot 2^n - 1 - s}{n\cdot 2^i} \right\rceil = 1 > \frac{n\cdot 2^n - 1 - s}{n\cdot 2^i}$$

which implies that

$$\begin{aligned} f(s) &= s + \sum_{i\le n} 2^i \left\lceil \frac{n\cdot 2^n - 1 - s}{n\cdot 2^i} \right\rceil \\ &> s + \sum_{i\le n} \frac{n\cdot 2^n - 1 - s}{n} = \sum_{i\le n} \frac{n\cdot 2^n - 1}{n} = n\cdot 2^n - 1 \end{aligned}$$

and from the integrality of both sides it follows that $f(s) \ge n \cdot 2^n$.

However, for $s = n \cdot 2^n - 1$ we have

$$f(s) = s + \sum_{i\le n} 2^i \left\lceil \frac{n\cdot 2^n - 1 - s}{n\cdot 2^i} \right\rceil = n\cdot 2^n - 1 < n\cdot 2^n$$

which reveals $\operatorname{lcm}_{i\le n} a_i - 1$ as the smallest optimal solution.

5.3.3 *Harmonic Periods*

Here we assume harmonic periods, i. e. $p_i \ge p_j$ implies $p_i/p_j \in \mathbb{Z}$ for all $i, j \in I$. A direct consequence is that $\operatorname{lcm}_{i\in I} p_i = \max_{i\in I} p_i$. Also it holds that $u \le u_1 \le p_{\max} \sum_{i\le n} c_i \le 2p_{\max}^2$. By using the described reduction to the Mixing Set instance ($\mathcal{M}$) and solving it in time $\mathcal{O}(n^2)$ using Theorem 16 we get the following result.

Theorem 18. *Let $I \subseteq [n-1]$ and $\gamma \in \mathbb{Z}_{\ge 1}$. If $k \ge \max_{i\in I} p_i$, then $\mathcal{R}(I,\gamma) \le_? k$ can be decided in time $\mathcal{O}(n^2)$.*

Obviously, by itself Theorem 18 does not allow to binary search for the optimal solution in general, since we do not know how to find it if it is smaller than $\max_{i\in I} p_i$. For now, we only see that we can utilize a binary search to get the following corollary.

Corollary 1. *Let t^* be the optimum value to ($\mathcal{R}$). If $t^* \ge \max_{i\in I} p_i$, then t^* can be computed in time $\mathcal{O}(n^2 \log p_{\max})$.*

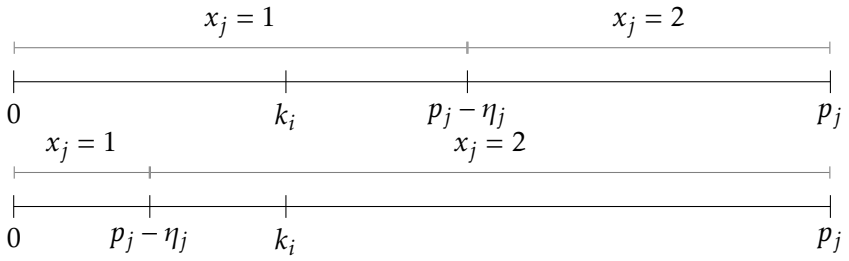


Figure 5.2: Two possible situations; the latter is more complicated

However, if the values for the variables of the tasks with the largest periods are known *a priori*, then the largest period of the residual instance will be smaller than before and we can decide even lower values of k. In fact, for tasks τ_i with large periods a rough knowledge about the optimum value suffices to find the exact value of the variables x_i. The following lemma will be crucial to exploit this idea (see also Fig. 5.2).

Lemma 37. *Let (t, x) be an optimal solution to ($\mathcal{R}$). If $0 < t \leq p_i$ for some $i \in I$, then*

$$x_i = \begin{cases} 1 & : t \leq p_i - \eta_i \\ 2 & : t > p_i - \eta_i \end{cases}.$$

Proof. Since (t, x) is an optimal solution, we know that $x_i = \lceil (t+\eta_i)/p_i \rceil$. Now, if we are in the case that $t \leq p_i - \eta_i$, then $0 < (t+\eta_i)/p_i \leq (p_i - \eta_i + \eta_i)/p_i = 1$ and if $p_i - \eta_i < t$, then $1 = (p_i - \eta_i + \eta_i)/p_i < (t + \eta_i)/p_i \leq (p_i + p_i)/p_i = 2$ which proves the claim. □

To describe our algorithm in detail, we consider integers $k_0 = 0$, $k_{q+1} = u$, as well as $0 < k_1 < \cdots < k_q$ with $\{k_1, \ldots, k_q\} = \{p_j - \eta_j \mid j \in I\} \setminus \{0\}$ where $q \leq |I|$. Our algorithm walks over these differences "from left to right", starting with the smallest difference $p_j - \eta_j$. Powered by the invariants Lemmas 38 and 39, it discovers variable values; in fact, the tasks whose variables are identified to have value 1 are stored in the set I_1 while those which are known to have value 2 are stored in set I_2.

Algorithm NARROW

1: **for** $i = 1, \ldots, q$ **do**
2: $\quad I_1 \leftarrow \{j \in I \mid k_i \leq p_j - \eta_j\}$
3: $\quad I_2 \leftarrow \{j \in I \mid p_j - \eta_j < k_i \leq p_j\}$
4: $\quad \gamma_i \leftarrow \gamma + \sum_{v \in \{1,2\}} \sum_{j \in I_v} v c_j$
5: $\quad$ **if** $\mathcal{M}(I \setminus (I_1 \cup I_2), k_i) \leq k_i - \gamma_i$ **then**
6: $\quad\quad$ **return** CATCH$(k_{i-1} + 1, k_i)$
7: **return** CATCH$(k_q + 1, u)$

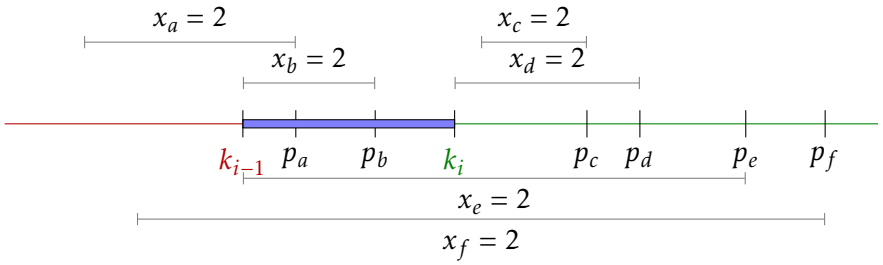


Figure 5.3: A (log-scaled) example where k_i is feasible while k_{i-1} is not, i. e. $k_{i-1} < \mathcal{R}(I, \gamma) \leq k_i$

Algorithm CATCH(L, R)

```
1:  I_1 ← {j ∈ I | R ≤ p_j − η_j}
2:  while L ≠ R do
3:      κ ← ⌊(L + R)/2⌋
4:      I_2 ← {j ∈ I | κ ≤ p_j < L + η_j}
5:      γ'_κ ← γ + Σ_{v∈{1,2}} Σ_{j∈I_v} v c_j
6:      if M(I \ (I_1 ∪ I_2), κ) ≤ κ − γ'_κ then
7:          R ← κ
8:      else
9:          L ← κ + 1
10: return R
```

So, the algorithm works in two steps. The first one is to identify the interval $(k_{i-1}, k_i]$ which contains the desired response time (Algorithm NARROW). The second step is to use a binary search inside this interval to find the exact response time (Algorithm CATCH). We heavily use the fact, that *any* solution which gives a YES answer to the decision problem $\mathcal{R}(I, \gamma) \leq_? k$ is smaller than k. The correctness of the algorithm crucially depends on the following two invariants.

Lemma 38 (Invariant). *For each iteration $i \in [q]$ of Algorithm* NARROW *and each $j \in I$ with $k_i \leq p_j$ the value for variable x_j is known a priori.*

Proof. We do a case distinction as follows. If $k_i \leq p_j - \eta_j$, then by Lemma 37 we know that $x_j = 1$. If otherwise $k_i > p_j - \eta_j$, then by the order of the algorithm we know that there is an index $i' < i$ such that we already have checked $k_{i'} = p_j - \eta_j < k_i$ and $k_{i'}$ was not feasible. Hence, any feasible solution t has to be truly greater than $k_{i'} = p_j - \eta_j$ and by Lemma 37 we know that $x_j = 2$. □

Lemma 39 (Invariant). *For each iteration $\kappa \in [L, R]$ of Algorithm* CATCH *and each $j \in I$ with $\kappa \leq p_j$ the value for variable x_j is known a priori.*

Proof. Remark that $k_i \notin [L, R)$ for each $i \in I$. We do a case distinction as follows. If $R \leq p_j$, then $x_j = 1$ if $R \leq p_j - \eta_j$ and $x_j = 2$ otherwise (if $p_j - \eta_j < R$

which implies $p_j - \eta_j \leq L$). If $\kappa \leq p_j < R$, then $p_j - \eta_j < L$ which implies $x_j = 2$. □

Figure 5.3 gives an example to see that in the second step all variable values for tasks with periods above κ are known a priori. The key property is, that by the definition of $k_1, \ldots, k_q$ there is no difference $p_j - \eta_j$ inside of the interval (k_{i-1}, k_i) for any task τ_j.

Lemma 40. *Let $i \in [q]$. If $\mathcal{R}(I,\gamma) > k_j$ for any $j < i$, then $\mathcal{R}(I,\gamma) \leq_? k_i$ can be decided in time $\mathcal{O}(n^2)$.*

Proof. Let us consider the following two task sets $I_1 = \{ j \in I \mid k_i \leq p_j - \eta_j \}$ and $I_2 = \{ j \in I \mid p_j - \eta_j < k_i \leq p_j \}$ and also $I_* = \{ j \in I \mid p_j < k_i \}$ such that $I_* = I \setminus (I_1 \dot\cup I_2)$. Then by using Lemma 38 and setting $\gamma' = \gamma + \sum_{v \in \{1,2\}, j \in I_v} v c_j$ the decision problem $\mathcal{R}(I,\gamma) \leq_? k_i$ simplifies to $\mathcal{R}(I_*,\gamma') \leq_? k_i$ as all variables for the tasks in I_1 have value 1 and the variables for the tasks in I_2 have value 2. By definition, all periods of the remaining tasks I_* are smaller than k_i, i. e. $k_i \geq \max_{j \in I_*} p_j$. Therefore, by using (5.6) again we can solve the Mixing Set problem $\mathcal{M}(I_*, k_i)$ to find the correct decision for $\mathcal{R}(I,\gamma) \leq_? k_i$ in time $\mathcal{O}(n^2)$. □

By Lemma 40 we see that Algorithm Narrow correctly narrows the response time candidate interval $(0, u]$ down to some interval $[L, R]$ such that $L = k_{i-1} + 1$ and $R = k_i$ for some $i \in [q+1]$. The correctness of the subsequent binary search in Algorithm Catch is then implied by Lemma 39. Together we get the following result.

Theorem 19. *Let $I \subseteq [n-1]$ and $\gamma \in \mathbb{Z}_{\geq 1}$. In the case of harmonic periods $\mathcal{R}(I,\gamma)$ can be computed in time $\mathcal{O}(q \cdot |I|^2 + |I|^2 \log p_{\max})$ where $q = |\{ p_i - \eta_i \mid i \in I \}|$.*

Note that the running time improves to strongly polynomial time $\mathcal{O}(q|I|^2)$ if $\log(p_{\max}) \leq \mathcal{O}(q)$ and to $\mathcal{O}(|I|^2 \log p_{\max})$ otherwise.

5.3.4 *Arbitrary Periods*

For the ease of notation and w.l.o.g. we consider $I = [n-1]$ and $\gamma = c_n$ here. The following theorem reveals a first simple algorithm to compute worst-case response times for general periods in pseudo-polynomial time.

Theorem 20. *The worst-case response time r_n can be computed in time $\mathcal{O}(n \cdot \operatorname{lcm}_{i<n} p_i)$.*

Proof. Let $m = \operatorname{lcm}_{i<n} p_i$ and define $w(t) = c_n + \sum_{i<n} c_i \lceil \frac{t+\eta_i}{p_i} \rceil$. Then there is a remainder $\rho \in \{0, \ldots, m-1\}$ and some $\lambda \in \mathbb{Z}_{\geq 0}$ such that $r_n = \rho + \lambda m$. We find that

$$\rho + \lambda m \;=\; r_n \;=\; w(r_n) \;=\; w(\rho + \lambda m) \;=\; w(\rho) + \lambda m \cdot \sum_{i<n} \frac{c_i}{p_i}$$

and thus, $\lambda = (w(\rho) - \rho) / ((1 - \sum_{i<n} \frac{c_i}{p_i}) \cdot m)$ can be computed in time $\mathcal{O}(n)$ for a given ρ. Hence, we can compute r_n by simply trying each ρ in $\{0, \ldots, m-1\}$

and compute λ for this choice to find the smallest sum $\rho + \lambda m$ for which the computed λ is a non-negative integer. This yields a total running time of $\mathcal{O}(n \cdot m)$. □

However, by using our reduction we can prove a Turing reduction as follows.

Theorem 21. *The response time r_n can be computed within a running time of $\mathcal{O}(\max(Sn, T_{\mathcal{M}} \log p_{\max})) \leq \mathcal{O}(Sn \log p_{\max})$ where S is a global upper bound on variable s in any optimal solution to ($\mathcal{M}$) and $T_{\mathcal{M}}$ is a global upper bound on the running time required to compute ($\mathcal{M}$).*

Proof. We may run an algorithm as follows. First decide $\mathcal{R}(I, \gamma) \leq_? S$. If the answer is YES, then try out all candidates $t \in [S-1]$ in time $\mathcal{O}(Sn)$, i. e. we check whether t is a solution to (5.1). If this reveals a feasible solution, return the smallest. Otherwise $r_n = S$. If the answer is NO, then it holds that $r_n > S$. From Lemma 32 we know that $u - \ell \leq p_{\max}^2$ and thus, a subsequent binary search in $[\ell, u]$ leads to a total running time of $\mathcal{O}(\max(Sn, T_{\mathcal{M}} \log(u-\ell))) \leq \mathcal{O}(\max(Sn, T_{\mathcal{M}} \log p_{\max}))$. For example we can bound $T_{\mathcal{M}}$ as follows. We can compute $\mathcal{M}([n-1], k)$ in time $\mathcal{O}(Sn)$ by simply trying solution $(s, x(s))$ for each $s \in [0, S)$. Hence, $T_{\mathcal{M}} \leq \mathcal{O}(Sn)$ and this gives a total running time of $\mathcal{O}(Sn \log p_{\max})$. □

In the case without task release jitter, i. e. $\eta_i = 0$ for all $i \in I$, we can even prove an *unconditional* Karp reduction as follows. For $\eta = 0$ we get the following form of ($\mathcal{M}$):

$$\begin{aligned} &\mathcal{M}_{\eta=0}(I, k) \\ &= \min\{ s + \textstyle\sum_{i \in I} c_i x_i \mid s + p_i x_i \geq k \ \forall i \in I,\ s \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^I \} \end{aligned} \qquad (\mathcal{M}_{\eta=0})$$

In this case we can show that $S \leq k$. Apparently, $(s = k, x = 0)$ is always a solution to ($\mathcal{M}_{\eta=0}$) since $s + p_i x_i = k$. It has the objective value $s + \sum_{i \in I} c_i x_i = k$. Any solution (s, x) to ($\mathcal{M}_{\eta=0}$) with $s > k$ has an objective value truly larger than k, since

$$\begin{aligned} s + \sum_{i \in I} c_i x_i(s) \ = \ s + \sum_{i \in I} c_i \left\lceil \frac{k-s}{p_i} \right\rceil \ &\geq \ s + \underbrace{(k-s)}_{<0} \cdot \underbrace{\sum_{i \in I} \frac{c_i}{p_i}}_{<1} \\ &> \ s + (k-s) \cdot 1 \ = \ k. \end{aligned}$$

Hence, any optimal solution (s, x) to ($\mathcal{M}_{\eta=0}$) holds $s \leq k$ which implies that $S \leq k$.

Therefore, $\overline{\mathcal{R}}(I, k) = k - \mathcal{M}(I, k)$ is directly implied. Thus, we can use a binary search to compute r_n in time $\mathcal{O}(T_{\mathcal{M}} \log p_{\max})$ where $T_{\mathcal{M}}$ is a global upper bound on the time required to compute $\mathcal{M}(I, k)$. This yields the following result.

Theorem 22. *If $\eta = 0$, then r_n can be computed in time $\mathcal{O}(T_{\mathcal{M}} \log p_{\max})$ where $T_{\mathcal{M}}$ is a global upper bound on the time required to compute ($\mathcal{M}$).*

SMALL UTILIZATION As for RM scheduling (cf. [94, 111]) the complexity of RTC decreases with a smaller utilization also for FP scheduling. We write $U = \sum_{i<n} c_i/p_i$ to denote the utilization.

Lemma 41. *In general it holds that* $S \le (\sum_{i<n} c_i)/(1-U)$. □

Proof. Let $S' = (\sum_{i<n} c_i)/(1-U)$, set $f(s) = s + \sum_{i<n} c_i x_i(s)$ as the objective function of $\mathcal{M}(I,k)$, and assume that $s \ge S'$ is a solution to $\mathcal{M}(I,k)$. Then

$$\begin{aligned} f(s-S') &= s - S' + \sum_{i<n} c_i \left\lceil \frac{k+\eta_i - s + S'}{p_i} \right\rceil \\ &\le s - S' + \sum_{i<n} c_i \left(\left\lceil \frac{k+\eta_i - s}{p_i} \right\rceil + \left\lceil \frac{S'}{p_i} \right\rceil \right) \\ &= f(s) - S' + \sum_{i<n} c_i \left\lceil \frac{S'}{p_i} \right\rceil \\ &< f(s) - S' + \sum_{i<n} c_i \left(\frac{S'}{p_i} + 1 \right) = f(s) \end{aligned}$$

which implies that $s' = s - S'$ is a truly better solution to $\mathcal{M}(I,k)$. □

Using the schedulability utilization bound this immediately assures that S is pseudo-polynomial using only the general utilization bound it suffices for example that $U \le 1 - p_{\max}^{-\mathcal{O}(1)}$. Since $u_1 - \ell = (\sum_{i<n} c_i)/(1-U)$ this also implies a pseudo-polynomial algorithm by simply guessing the right value in in the interval $[\ell, u]$. However, especially if $r_n > S$ our algorithm may be faster.

5.4 MIXING SET

Our technique may be reversed to solve the MIXING SET problem by computing worst-case response times. This allows us to make efficient approaches to RTC applicable to solve MIXING SET. We simply do a substitution in (5.6) (substitute $k \mapsto k' + \gamma$, $\gamma \mapsto \beta - k'$, $k' \mapsto k$ in this order) to achieve the following corollary.

Corollary 2. *Let* $\beta \ge S$. *Then* $\mathcal{M}(I,\beta) \le k$ *if and only if* $\mathcal{R}(I, \beta - k) \le \beta$.

With the reverse reduction we can improve the running time for general capacities, if the entries of the right-hand side b are large and close to $b_{\min}$:

Lemma 42. *Given capacities* $a \in \mathbb{Z}^n_{\ge 1}$, *a right-hand side* $b \in \mathbb{Z}^n$ *which holds that* $\operatorname{lcm}_{j\le n} a_j \le b_i \le b_{\min} + a_i$ *for all* $i \le n$, *and weights* $w \in \mathbb{Z}^n_{\ge 0}$, *one can compute*

$$\min\{ s + \sum_{i\le n} w_i x_i \mid s + a_i x_i \ge b_i \ \forall i \le n,\ s \in \mathbb{Z}_{\ge 0},\ x \in \mathbb{Z}^n \} \qquad (\mathcal{M}_b)$$

in time $\mathcal{O}(T_{\mathcal{R}} \log b_{\max})$ *where* $T_{\mathcal{R}}$ *is a global upper bound on the time required to compute* ($\mathcal{R}$).

Proof. Choose $\beta = b_{\min}$ and $\eta_i = b_i - b_{\min}$ for all $i \in [n]$. We get $\beta \geq \operatorname{lcm}_{i \leq n} a_i \geq S$ and $0 \leq \eta_i \leq a_i$ for all $i \leq n$. This implies $b_i = \beta + \eta_i$ and thus, we can write ($\mathcal{M}_b$) as

$$\min \{ s + \sum_{i \leq n} w_i x_i \mid s + a_i x_i \geq \beta + \eta_i \ \forall i \leq n,\ s \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^n \}.$$

Now, we use Corollary 2 to start a binary search for the optimal k. We know that $\mathcal{R}([n], \beta - k)$ can be computed in time $T_{\mathcal{R}}$ and the objective value of our Mixing Set instance is upper bounded by $b_{\max}$ (since $(s = b_{\max}, x = 0)$ is always a solution with objective value $b_{\max}$) which implies a total running time of $\mathcal{O}(T_{\mathcal{R}} \log b_{\max})$. □

In fact, Lemma 42 implicitly solves the general case. We use it to prove the following.

Theorem 23. *Given capacities $a \in \mathbb{Z}^n_{\geq 1}$, a right-hand side $b \in \mathbb{Z}^n$, and weights $w \in \mathbb{Z}^n_{\geq 0}$, one can compute*

$$\min \{ s + \sum_{i \leq n} w_i x_i \mid s + a_i x_i \geq b_i \ \forall i \leq n,\ s \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^n \} \tag{$\mathcal{M}_b$}$$

in time $\mathcal{O}(n + T_{\mathcal{R}} \log b_{\max})$ where $T_{\mathcal{R}}$ is a global upper bound on the time required to compute ($\mathcal{R}$).

Proof. Let $m = \operatorname{lcm}_{j \leq n} a_j$. We define a new right-hand side $b' \in \mathbb{Z}^n$ by

$$b'_i = b_i + \left\lceil \frac{m - b_i}{a_i} \right\rceil a_i$$

for each $i \leq n$. It is easy to see that $m \leq b'_i \leq m + a_i \leq b'_{\min} + a_i$ for all $i \leq n$. Also, we have

$$\begin{aligned}
&\min \{ s + \sum_{i \leq n} w_i x_i \mid s + a_i x_i \geq b'_i \forall i \leq n,\ s \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^n \} \\
&= \min \{ s + \sum_{i \leq n} w_i \left\lceil \frac{b'_i - s}{a_i} \right\rceil \mid s \in \mathbb{Z}_{\geq 0} \} \\
&= \min \{ s + \sum_{i \leq n} w_i \left\lceil \frac{b_i + \left\lceil \frac{m - b_i}{a_i} \right\rceil a_i - s}{a_i} \right\rceil \mid s \in \mathbb{Z}_{\geq 0} \} \\
&= \sum_{i \leq n} w_i \left\lceil \frac{m - b_i}{a_i} \right\rceil + \min \{ s + \sum_{i \leq n} w_i \left\lceil \frac{b_i - s}{a_i} \right\rceil \mid s \in \mathbb{Z}_{\geq 0} \} \\
&= \sum_{i \leq n} w_i \left\lceil \frac{m - b_i}{a_i} \right\rceil \\
&\qquad + \min \{ s + \sum_{i \leq n} w_i x_i \mid s + a_i x_i \geq b_i \forall i \leq n,\ s \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^n \}
\end{aligned}$$

Hence, we can solve the system for b' with Lemma 42 and simply subtract the sum $\sum_{i \leq n} w_i \lceil (m - b_i)/a_i \rceil$ to obtain the optimal value. This gives the desired running time. □

Now, we aim to identify the instances of Mixing Set which may be solved by using our reversed reduction in combination with efficient algorithms for the computation of worst-case response times. By turning to harmonic periods p_i and using Corollary 2 again, we can apply Nguyen et al. [102] to compute

$$\min\{s + \sum_{i\in I} c_i x_i \mid s + p_i x_i \geq \beta \ \forall i \in I,\ s \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^I\}$$

in time $\mathcal{O}(n \log n + n \log \beta)$ for any $\beta \geq \max_{i\in I} p_i$ by computing the response times

$$\min\{t \mid t \geq \beta - k + \sum_{i\in I} \left\lceil \frac{t}{p_i} \right\rceil c_i,\ t \in \mathbb{Z}_{\geq 0}\}$$

in a binary search for the smallest feasible k. It suffices to search in the interval $[0, \beta]$ since $(s = \beta, x = 0)$ is always a solution with value β. Remark that the usual running time to solve instances of Mixing Set with harmonic capacities is $\mathcal{O}(n^2)$. In a more classical formulation this yields the following theorem.

Theorem 24. *Given harmonic capacities $a \in \mathbb{Z}^n_{\geq 1}$, a right-hand side value $\beta \geq a_{\max}$, and weights $w \in \mathbb{Z}_{\geq 0}$, one can compute*

$$\min\{s + \sum_{i\leq n} w_i x_i \mid s + a_i x_i \geq \beta \ \forall i \leq n,\ s \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^n\} \tag{$\mathcal{M}_\beta$}$$

in time $\mathcal{O}(n \log n + n \log \beta) \leq \mathcal{O}(n \log(n + \beta))$.

The algorithm of Nguyen et al. runs in linear time $\mathcal{O}(n)$ after an initial sorting step in time $\mathcal{O}(n \log n)$ to sort the tasks by their periods. As we seek to apply the algorithm multiple times, we can save the time to sort by only doing it once.

However, for arbitrary capacities and the special case $b_i = \beta$ for all $i \leq n$ and $\beta \geq \operatorname{lcm}_{j\leq n} a_j$ we can also derive the following corollary from Lemma 42.

Corollary 3. *Given capacities $a \in \mathbb{Z}^n_{\geq 1}$, a right-hand side value $\beta \geq \operatorname{lcm}_{i\leq n} a_i$, and weights $w \in \mathbb{Z}^n_{\geq 0}$, one can compute*

$$\min\{s + \sum_{i\leq n} w_i x_i \mid s + a_i x_i \geq \beta \ \forall i \leq n,\ s \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^n\} \tag{$\mathcal{M}_b$}$$

in time $\mathcal{O}(T_{\mathcal{R}} \log \beta)$ where $T_{\mathcal{R}}$ is a global upper bound on the time required to compute ($\mathcal{R}$).

5.5 Hardness of approximation of 4-block integer programs

Eisenbrand and Rothvoß [45] proved that worst-case response times cannot be approximated to a constant factor, unless P = NP, even if the task system has a total utilization truly smaller than 1. Here we reduce response time

computation to a 4-block integer linear program. The worst-case response time of a task system without jitters is

$$r_n = \min\{\, t \mid t \geq c_n + \sum_{i=1}^{n-1} \left\lceil \frac{t}{p_i} \right\rceil c_i,\ t \in \mathbb{Z}_{\geq 0} \,\}$$

which may be restated by the following formulation as an integer linear program:

$$r_n = \min\{\, t \mid t \geq c_n + \sum_{i=1}^{n-1} x_i c_i,\ p_i x_i \geq t\ \forall i \in [n-1],\ t \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^{n-1} \,\}.$$

Thus, we can introduce a 4-block matrix $\mathcal{A} \in \mathbb{Z}^{n \times n}$ and a right-hand side $b \in \mathbb{Z}^n_{\geq 0}$ leading to

$$r_n = \min\{\, t \mid \mathcal{A} \begin{pmatrix} t \\ x \end{pmatrix} \geq b,\ t \in \mathbb{Z}_{\geq 0},\ x \in \mathbb{Z}^{n-1} \,\}$$

where

$$\mathcal{A} = \left(\begin{array}{c|ccc} 1 & -c_1 & \cdots & -c_{n-1} \\ \hline -1 & p_1 & & \\ \vdots & & \ddots & \\ -1 & & & p_{n-1} \end{array}\right) \quad \text{and} \quad b = \begin{pmatrix} c_n \\ \hline 0 \\ \vdots \\ 0 \end{pmatrix}.$$

As $p_i x_i \geq t \geq 0$ and $p_i \geq 1$ imply that $x_i \geq 0$ this leads to the following formulation as a 4-block integer program:

$$r_n = \min\{\, x_1 \mid \mathcal{A}x \geq b,\ x \geq 0,\ x \in \mathbb{Z}^n \,\}$$

Therefore, it is NP-hard to approximate 4-block ILP to a constant factor, even if all block matrices have size 1×1, all constraints are inequalities, the objective function addresses the first variable only, the variables are unbounded, and the right-hand side has at most one non-zero entry.

5.6 4-BLOCK INTEGER PROGRAMMING

Block-structured integer programming has received major interest in the recent past. Namely, the three most important lines of research are n-fold integer programming, 2-stage stochastic integer programming, and 4-block integer programming [43, 29, 104, 62]. The latter is the most general but also the least understood flavor of block-structured integer programs. In this section we will consider 4-block integer programs with only one coupling constraint inequality. Furthermore, we consider a simpler objective function which addresses at most two *bricks*. From Section 5.5 it follows that this problem is still hard to approximate to any constant factor. In fact, the literature on 4-block integer programs is not unified in its formal presentation. Here we consider 4-block integer programs as follows.

Consider block matrices $A_1,\dots,A_n$, $B_1,\dots,B_n$, $C_1,\dots C_n$, and D such that $A_i \in \mathbb{Z}^{r\times t}$, $B_i \in \mathbb{Z}^{r\times s}$, and $C_i \in \mathbb{Z}^{q\times t}$ for all $i \in [n]$ and $D \in \mathbb{Z}^{q\times s}$. We aim to compute

$$\min\{\, w^\intercal x \mid \mathcal{B}x = b,\ 0 \le x \le u,\ x \in \mathbb{Z}^{s+nt} \,\} \tag{5.7}$$

where

$$\mathcal{B} = \begin{pmatrix} D & C_1 & \cdots & C_n \\ B_1 & A_1 & & \\ \vdots & & \ddots & \\ B_n & & & A_n \end{pmatrix}.$$

We use the notion of *bricks* $w^{(0)}, w^{(1)}, \dots, w^{(n)}$ and $x^{(0)}, x^{(1)}, \dots, x^{(n)}$ to refer to different areas of w and x. In more detail, for each $i \in \{1,\dots,n\}$ we define $w^{(i)} = (w_{(i-1)t+1},\dots,w_{it})$ and $x^{(i)} = (x_{(i-1)t+1},\dots,x_{it})$ to denote the i-th *brick* of w and x, respectively. Furthermore, let $w^{(0)} = (w_1,\dots,w_s)$ and $x^{(0)} = (x_1,\dots,x_s)$ denote the 0-th brick of w and x, respectively.

In the following we assume that there is exactly one coupling constraint *inequality* and that there is a $j \in [n]$ such that $w^{(i)} = 0$ for all $i \in [n] \setminus \{j\}$. In other words, the objective function addresses exactly two bricks of the solution and at least one of them is the first brick. Let $a \in \mathbb{Z}^{s+nt}$ denote the first row of $\mathcal{B}$. We aim to compute

$$\tau = \min\{\, w^\intercal x \mid a^\intercal x \ge b_0,\ \mathcal{B}'x = b,\ 0 \le x \le u,\ x \in \mathbb{Z}^{s+nt} \,\} \tag{5.8}$$

where

$$\mathcal{B}' = \begin{pmatrix} C_1 & B_1 & & \\ \vdots & & \ddots & \\ C_n & & & B_n \end{pmatrix}.$$

For the decision problem $\tau \le_? k$ we get the following dual formulation:

$$\max\{\, a^\intercal x \mid w^\intercal x \le k,\ \mathcal{B}'x = b,\ 0 \le x \le u,\ x \in \mathbb{Z}^{s+nt} \,\} \ge_? b_0$$

Introduce a slack variable $y \ge 0$ to replace the inequality "$w^\intercal x \le k$" with the equation "$w^\intercal x + y = k$":

$$\max\{\, a^\intercal x \mid w^\intercal x + y = k,\ \mathcal{B}'x = b,\ 0 \le x \le u,\ x \in \mathbb{Z}^{s+nt},\ y \in \mathbb{Z}_{\ge 0} \,\}$$

By placing the additional constraint "$w^\intercal x + y = k$" on top of the j-th block matrices we get a new matrix

$$\mathcal{B}'' = \begin{pmatrix} C_1 & B_1 & & \\ \overline{w}^{(0)} & & \overline{w}^{(j)} & 1 \\ C_j & & B_j & \\ & & & \\ C_n & & & B_n \end{pmatrix}$$

where $\overline{w}^{(0)} = (w^{(0)})^\intercal$ and $\overline{w}^{(j)} = (w^{(j)})^\intercal$. Finally, we arrive at the 2-stage stochastic integer program

$$\max\{a^\intercal x \mid \mathcal{B}'' x = b',\ 0 \le x \le u,\ x \in \mathbb{Z}^{s+nt},\ y \in \mathbb{Z}_{\ge 0}\} \tag{5.9}$$

where b' is the modified right-hand side with value k. Use, for example, [35] to solve it.

Theorem 25. *Program* (5.8) *can be solved in time* $\mathcal{O}(T_{(5.9)} \cdot \log(2H))$ *where* $T_{(5.9)}$ *is a global upper bound on the time to solve* (5.9) *and* H *is an upper bound on the absolute objective value of an optimal solution to* (5.8).

Using Theorem 2 we can bound $T_{(5.9)} \le 2^{(2\Delta)^{\mathcal{O}(s(s+t))}} \cdot n \log^{\mathcal{O}(st)} n$ by using parallel computation where $\Delta = \max\{\|\mathcal{B}\|_\infty, \|w\|_\infty\}$ is the largest absolute value of the entries in $\mathcal{B}$ and w. To give a simple bound on H one can use standard arguments (e. g. [107, Theorem 13.5]) to show that

$$\log(2H) \le \mathcal{O}(\log(st\|w\|_1\|b\|_\infty) + (nr+q)\log((nr+q)\|A\|_\infty)).$$

5.7 OPEN QUESTIONS

Although we have achieved a better understanding of response time computation in real-time task systems, the presented results raise a number of questions.

On the one hand our algorithm for harmonic periods is very efficient and allows an implementation in practice but on the other hand it is unclear whether its running time may be improved even further. Can we prove a lower bound on the running time?

Also, the investigation of *arbitrary* deadlines [17, 90] seems to be more complicated and it would be interesting to find appropriate generalizations of our algorithms for this extended setting as well.

The complexity result of Ekberg [48] does not consider task release jitters. Therefore, a natural question is: Where are the utilization borders between polynomial and NP-hard schedulability testing in the presence of task release jitters?

6 FUZZY SIMULTANEOUS CONGRUENCES

6.1 INTRODUCTION

In this chapter we present a generalization of MIXING SET (see Chapter 5 and especially Section 5.2.2) which simultaneously generalizes the problem of simultaneous congruences. Remember that MIXING SET is a special case of a 2-stage stochastic ILP.

A result in the field of real-time systems by Nguyen, Grass, and Jansen [102] (cf. Chapter 5) gives rise to the study of a new problem. Their algorithm uses heuristic components to solve an integer program that can be stated as a bounded version of MIXING SET with additional upper bounds B_i as follows.

BOUNDED MIXING SET (BMS)
Given capacities $a_1, \dots, a_n \in \mathbb{Z}$ and bounds $b, B \in \mathbb{Z}^n$ find $(s, x) \in \mathbb{Z}_{\geq 0} \times \mathbb{Z}^n$ such that

$$b_i \leq s + a_i x_i \leq B_i \quad \forall i \in [n].$$

In particular they depend on minimizing the value of s which can be achieved in linear time in case of MIXING SET. While BMS may look artificial at first sight, it is not; in fact, leading to a very natural generalization it can be restated in the well-known form of *simultaneous congruences*.

FUZZY SIMULTANEOUS CONGRUENCES (FSC)
Given divisors $a_1, \dots, a_n \in \mathbb{Z} \setminus \{0\}$ and remainder intervals $R_1, \dots, R_n \subseteq \mathbb{Z}$ and an interval $S \subseteq \mathbb{Z}_{\geq 0}$ find a number $s \in S$ such that

$$\exists\, r_i \in R_i : s \equiv r_i \pmod{a_i} \quad \forall i \in [n].$$

Obviously, this also generalizes over the well-known problem of the Chinese Remainder Theorem (CRT). Here we give its generalized form (cf. [87]).

Theorem 26 (Generalized Chinese Remainder Theorem)**.** *Given n distinct divisors $a_1, \dots, a_n \in \mathbb{Z}_{\geq 1}$ and n remainders $r_1, \dots, r_n \in \mathbb{Z}_{\geq 0}$ the system of n simultaneous congruences $s \equiv r_i \pmod{a_i}$ admits a solution $s \in \mathbb{Z}$ if and only if $r_i \equiv r_j \pmod{\gcd(a_i, a_j)}$ for all $i \neq j$.*

Furthermore, Leung and Whitehead [92] showed that k-Simultaneous Congruences (k-SC) is NP-complete in the weak sense. Given divisors $a_1, \dots, a_n \in \mathbb{Z}_{\geq 1}$ and remainders $r_1, \dots, r_n \in \mathbb{Z}_{\geq 0}$ the task is to find a number $s \in \mathbb{Z}_{\geq 0}$ and a subset $I \subseteq \{1, \dots, n\}$ with $|I| = k$ s.t. $s \equiv r_i \pmod{a_i}$ for all $i \in I$. Later it was shown by Baruah et al. [14] that k-SC also is NP-complete in the strong sense.

Both problems BMS and FSC are interchangeable formulations of the same problem (see Section 6.2). Therefore, we will use them as synonyms and we especially assume formally that $R_i = [b_i, B_i]$. Interestingly and to the best

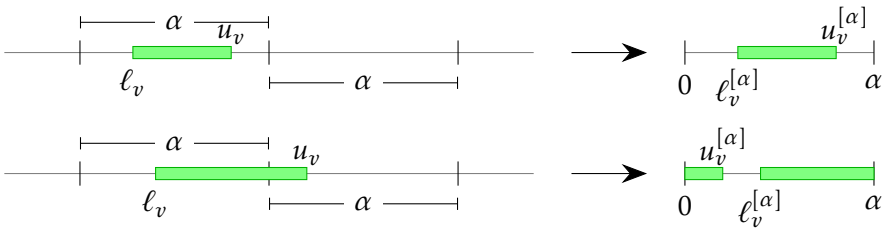


Figure 6.1: The two possibilities for the modular projection of an interval

of our knowledge, FSC/BMS was not considered before. However, the investigation of simultaneous congruences has always been of transdisciplinary interest connecting a variety of fields and applications, e. g. [6, 52, 60].

OUR CONTRIBUTION

(a) We show that BMS is NP-hard for general capacities a_i. For the reduction from DIRECTED DIOPHANTINE APPROXIMATION we refer to the appendix. Compared to MIXING SET this is a stronger hardness result as BMS by itself only asks for an *arbitrary* feasible solution. Remark that every feasible instance of MIXING SET may be solved by $s = \|b\|_\infty, x = \mathbf{0}$.

(b) In the case of harmonic capacities (i. e. a_{i+1}/a_i is an integer for all $i < n$), which was heavily studied for MIXING SET as mentioned before, we give an algorithm exploiting a merge idea based on modular arithmetic on intervals to decide the feasibility problem of FSC in time $\mathcal{O}(n^2)$. See Section 6.3.1 for the details.

(c) Furthermore, for a feasible instance of FSC with harmonic capacities we present a polynomial algorithm as well as a strongly polynomial algorithm to compute the smallest feasible solution to FSC in time $\mathcal{O}(\min\{n^2 \log(a_n), n^3\}) \leq \mathcal{O}(n^3)$. See Section 6.3.2 for the details.

6.2 PRELIMINARIES

For the sake of readability we write $X^{[\alpha]} = (X \bmod \alpha)$ for numbers X as well as $X^{[\alpha]} = \{z \bmod \alpha \mid z \in X\}$ for sets X (of numbers) to denote the modular projection of some number or interval, respectively. Extending the usual notation we also write $X \equiv Y \pmod{\alpha}$ if $X^{[\alpha]} = Y^{[\alpha]}$ for sets X, Y. Notice that on the one hand $(X \cup Y)^{[\alpha]} = X^{[\alpha]} \cup Y^{[\alpha]}$ but on the other hand be aware that $(X \cap Y)^{[\alpha]} \neq X^{[\alpha]} \cap Y^{[\alpha]}$ in general (cf. Lemma 46). Figure 6.1 depicts the structure of $v^{[\alpha]}$ if $v = [\ell_v, u_v]$ is an interval in $\mathbb{Z}$.

Also we use the well-tried notation $t + X = \{t + z \mid z \in X\}$ to express the *translation* of a set of numbers X by some number t. For a set of sets $\mathcal{S}$ we write $\bigcup \mathcal{S}$ to denote the union $\bigcup_{S \in \mathcal{S}} S$. Furthermore, we identify constraints by their indices. So for $i \leq n$ we say that "$b_i \leq s + a_i x_i \leq B_i$" *is* constraint i.

IDENTITY OF BMS AND FSC

It is important to notice that BMS allows zero capacities while FSC cannot allow zero divisors since (mod 0) is undefined.

However, consider a constraint i of BMS with $a_i \neq 0$. Let $b_i \leq s + a_i x_i \leq B_i$ be satisfied and set $r_i = s + a_i x_i$. Then $r_i^{[a_i]} = s^{[a_i]}$ and $r_i \in [b_i, B_i] = R_i$. Vice-versa let $r_i \in R_i$ s.t. $r_i \equiv s \pmod{a_i}$. Then there is an $x_i \in \mathbb{Z}$ such that $s + a_i x_i = r_i \in R_i = [b_i, B_i]$.

A constraint i that holds $a_i = 0$ simply demands that $s \in R_i$. Hence, if $a_i = a_j = 0$ for two constraints $i \neq j$ they can be replaced by one new constraint k defined by $R_k = R_i \cap R_j$. Therefore, one may assume that there is at most one constraint i with a zero capacity a_i. However, as all our results can be lifted back to the general case with low effort we will assume in terms of BMS that all capacities are non-zero and for FSC we make the equivalent assumption that $S = \mathbb{Z}_{\geq 0}$.

With our notation we may easily express the feasibility of a value s for a single constraint i as follows.

Observation 15. *A value s satisfies constraint i if and only if $s^{[a_i]} \in R_i^{[a_i]}$.*

Proof. $\exists r_i \in R_i : r_i \equiv s \pmod{a_i}$ iff $\exists r_i \in R_i : r_i^{[a_i]} = s^{[a_i]}$ iff $s^{[a_i]} \in R_i^{[a_i]}$. □

By simply swapping the signs of the x_i we may assume that $a_i \geq 0$ for all i. We may also assume that the intervals are small in the sense that $B_i - b_i + 1 < a_i$ holds for all i. Assume that $B_i - b_i + 1 \geq a_i$ for an i and let $s \geq 0$ be an arbitrary integer. Then $b_i \leq B_i - a_i + 1$ and constraint i may always be solved by setting $x_i = \lceil (b_i - s)/a_i \rceil$ which satisfies

$$b_i \leq s + a_i \underbrace{\lceil \tfrac{b_i - s}{a_i} \rceil}_{x_i} \leq s + a_i \lceil \tfrac{B_i - a_i + 1 - s}{a_i} \rceil = s + a_i \lfloor \tfrac{B_i - s}{a_i} \rfloor \leq B_i.$$

Hence, constraint i is redundant and may be omitted. As a direct consequence there can be at most one feasible value for each x_i for a given guess s. In fact, we can decide the feasibility of a guess s in time $\mathcal{O}(n)$ as for all constraints i and values x_i it holds $b_i \leq s + a_i x_i \leq B_i$ if and only if $\lceil (b_i - s)/a_i \rceil = x_i = \lfloor (B_i - s)/a_i \rfloor$. So a guess s is feasible if and only if $\lceil (b_i - s)/a_i \rceil = \lfloor (B_i - s)/a_i \rfloor$ holds for all constraints i. Another consequence is that BMS is a generalization of MIXING SET as one can always add trivial upper bounds. By $s_{\min}$ we denote the smallest feasible solution s that satisfies all constraints.

Observation 16. *For feasible instances it holds that* $s_{\min} < \operatorname{lcm}(a_1, \ldots, a_n)$.

Proof. Let $\varphi = \operatorname{lcm}(a_1, \ldots, a_n)$. Remark that φ/a_i is integral for all i. Assume that (s, x) is a solution with $s = s_{\min} \geq \varphi$. Let $t = s - \varphi$ and $y_i = x_i + \varphi/a_i$ f.a. i. Then $0 \leq t < s_{\min}$ and $t + a_i y_i = s + a_i x_i$ f.a. i. So (t, y) is a solution that contradicts the optimality. □

6.3 HARMONIC DIVISORS

Here we consider harmonic divisors in the sense that a_{i+1}/a_i is an integer for all $i < n$.

As we investigate some kind of a generalization of the setting of the Chinese Remainder Theorem, it is natural to ask for a CRT for *harmonic* (instead of the usually coprime) divisors and of course the (generalized) CRT answers

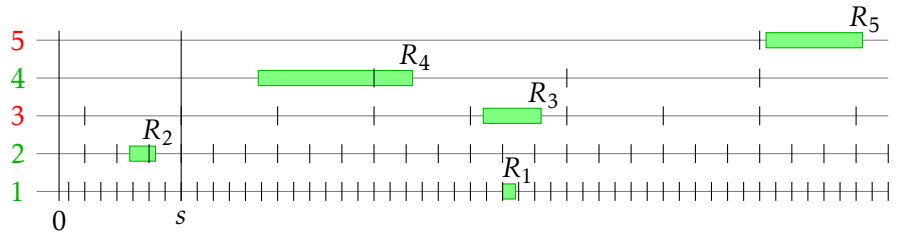


Figure 6.2: $36a_1 = 18a_2 = 6a_3 = 3a_4 = a_5$. The guess s is not feasible for constr. 3 and 5

this question; in this case we have $\gcd(a_i, a_n) = a_i$ and so Theorem 26 reveals that if the system of n simultaneous congruences $s \equiv r_i \pmod{a_i}$ admits a solution then $r_i \equiv r_n \pmod{a_i}$ which says that if there is any solution then the set of all solutions is $a_n\mathbb{Z} + r_n^{[a_n]}$. However, it turns out that the investigation of FSC is a lot more complicated.

In this section we present an algorithm to decide the feasibility of an instance of FSC. Also we show how optimal solutions can be computed in (strongly) polynomial time. Both of these results are based on the fine-grained interconnection between modular arithmetic on sets and the harmonic property. For some intuition Figure 6.2 gives a perspective on s as an anchor for 1-dimensional lattices with basis a_i which have to "hit" the intervals R_i. For example, in the figure it holds that $s + a_2 \cdot (-1) = s - a_2 \in R_2$, so the 1-dimensional lattice $(s + a_2 z)_{z\in\mathbb{Z}}$ hits interval R_2. Therefore, the choice of s satisfies constraint 2.

6.3.1 *Deciding feasibility*

The idea for our first algorithm will be to decide the feasibility problem by iteratively computing modular projections from constraint $i = n$ down to $i = 1$. In the following we will say that an interval $w \subseteq \mathbb{Z}$ *represents* a set $M \subseteq \mathbb{Z}$ (modulo α) if $w^{[\alpha]} = M^{[\alpha]}$. Also a set of intervals $\mathcal{R}$ represents a set $M \subseteq \mathbb{Z}$ (modulo α) if $M^{[\alpha]} = \bigcup_{w\in\mathcal{R}} w^{[\alpha]}$. Given an integer $\alpha \geq 1$ and two intervals v, w we need to study the structure of the intersection $v^{[\alpha]} \cap w^{[\alpha]} \subseteq [0, \alpha)$. To express it let $v = [\ell_v, u_v], w = [\ell_w, u_w]$ and we define the basic intervals

$$\varphi_\alpha(v, w) = [\ell_v^{[\alpha]}, u_w^{[\alpha]}]$$

and

$$\psi_\alpha(v, w) = [\max\{\ell_v^{[\alpha]}, \ell_w^{[\alpha]}\}, \alpha + \min\{u_v^{[\alpha]}, u_w^{[\alpha]}\}].$$

for all intervals v, w. The former may be thought as the cases where $v^{[\alpha]}$ and $w^{[\alpha]}$ are two overlapping intervals while the intuition for the latter are situations where $v^{[\alpha]}$ and $w^{[\alpha]}$ both consist of two intervals which are in pairs overlapping. Remark that $\psi_\alpha(w, v) = \psi_\alpha(v, w)$ is always true.

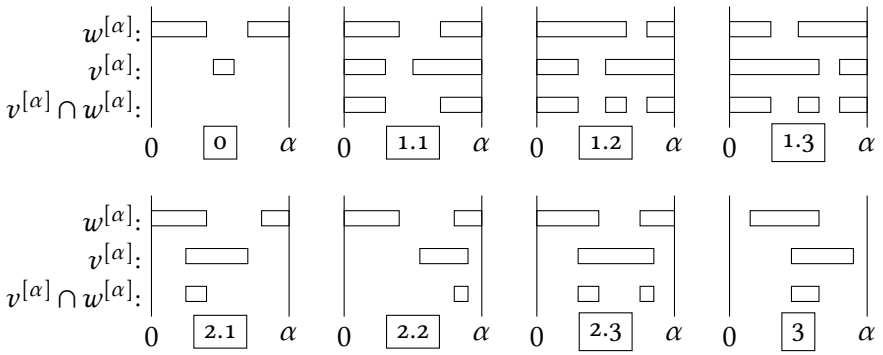


Figure 6.3: Examples for the cases of the case distinction in the proof of Lemma 43

Lemma 43. *Given an integer $\alpha \geq 1$ and two intervals $v, w \subseteq \mathbb{Z}$ it holds that the intersection $v^{[\alpha]} \cap w^{[\alpha]}$ is in the set*

$$\begin{aligned}
&\{\, \varnothing,\ v^{[\alpha]},\ w^{[\alpha]},\ \psi_\alpha(v,w)^{[\alpha]},\ \varphi_\alpha(v,w),\ \varphi_\alpha(w,v),\\
&\quad \varphi_\alpha(v,w) \dot\cup \varphi_\alpha(w,v),\\
&\quad \varphi_\alpha(v,w) \dot\cup \psi_\alpha(v,w)^{[\alpha]},\\
&\quad \varphi_\alpha(w,v) \dot\cup \psi_\alpha(v,w)^{[\alpha]} \,\}.
\end{aligned}$$

Proof. We do a case distinction (see Figure 6.3) as follows. We only look at the non-trivial case, i. e. $v^{[\alpha]} \cap w^{[\alpha]} \notin \{\varnothing, v^{[\alpha]}, w^{[\alpha]}\}$, which especially implies $|v| < \alpha$ and $|w| < \alpha$.

We start with the case that neither $v^{[\alpha]}$ nor $w^{[\alpha]}$ is an interval, i. e. $u_v^{[\alpha]} < \ell_v^{[\alpha]}$ and $u_w^{[\alpha]} < \ell_w^{[\alpha]}$. Then it cannot be that $u_w^{[\alpha]} \geq \ell_v^{[\alpha]}$ and $u_v^{[\alpha]} \geq \ell_w^{[\alpha]}$ since that implies $\ell_v^{[\alpha]} \leq u_w^{[\alpha]} < \ell_w^{[\alpha]} \leq u_v^{[\alpha]}$. Hence, there are three cases as follows.
Case 1.1. $u_w^{[\alpha]} < \ell_v^{[\alpha]}$ and $u_v^{[\alpha]} < \ell_w^{[\alpha]}$. Then the intersection equals

$$\begin{aligned}
&[0, \min\{u_v^{[\alpha]}, u_w^{[\alpha]}\}] \dot\cup [\max\{\ell_v^{[\alpha]}, \ell_w^{[\alpha]}\}, \alpha)\\
&= [\max\{\ell_v^{[\alpha]}, \ell_w^{[\alpha]}\}, \alpha + \min\{u_v^{[\alpha]}, u_w^{[\alpha]}\}]^{[\alpha]} = \psi_\alpha(v,w)^{[\alpha]}.
\end{aligned}$$

Case 1.2. $u_w^{[\alpha]} \geq \ell_v^{[\alpha]}$ and $u_v^{[\alpha]} < \ell_w^{[\alpha]}$. Then the intersection equals

$$\begin{aligned}
[0, u_v^{[\alpha]}] \dot\cup [\ell_v^{[\alpha]}, u_w^{[\alpha]}] \dot\cup [\ell_w^{[\alpha]}, \alpha) &= [\ell_v^{[\alpha]}, u_w^{[\alpha]}] \dot\cup [\ell_w^{[\alpha]}, \alpha + u_v^{[\alpha]}]^{[\alpha]}\\
&= \varphi_\alpha(v,w) \dot\cup \psi_\alpha(v,w)^{[\alpha]}.
\end{aligned}$$

Case 1.3. $u_w^{[\alpha]} < \ell_v^{[\alpha]}$ and $u_v^{[\alpha]} \geq \ell_w^{[\alpha]}$. By simply using symmetry we get $v^{[\alpha]} \cap w^{[\alpha]} = \varphi_\alpha(w,v) \dot\cup \psi_\alpha(v,w)^{[\alpha]}$.

Now, w.l.o.g. assume that $v^{[\alpha]}$ is an interval, i. e. $\ell_v^{[\alpha]} \leq u_v^{[\alpha]}$, while $w^{[\alpha]}$ consists of two intervals, i. e. $u_w^{[\alpha]} < \ell_w^{[\alpha]}$. Then there are three cases as follows.
Case 2.1. $\ell_v^{[\alpha]} \leq u_w^{[\alpha]} < u_v^{[\alpha]} < \ell_w^{[\alpha]}$. The intersection is $[\ell_v^{[\alpha]}, u_w^{[\alpha]}] = \varphi_\alpha(v,w)$.
Case 2.2. $u_w^{[\alpha]} < \ell_v^{[\alpha]} < \ell_w^{[\alpha]} \leq u_v^{[\alpha]}$. The intersection is $[\ell_w^{[\alpha]}, u_v^{[\alpha]}] = \varphi_\alpha(w,v)$.

Case 2.3. $\ell_v^{[\alpha]} \le u_w^{[\alpha]} < \ell_w^{[\alpha]} \le u_v^{[\alpha]}$. Then the intersection is

$$[\ell_v^{[\alpha]}, u_w^{[\alpha]}] \dot\cup [\ell_w^{[\alpha]}, u_v^{[\alpha]}] = \varphi_\alpha(v,w) \dot\cup \varphi_\alpha(w,v).$$

Clearly, if both $v^{[\alpha]}$ and $w^{[\alpha]}$ are intervals (Case 3) (which are not disjoint) then their intersection is either $\varphi_\alpha(v,w)$ or $\varphi_\alpha(w,v)$. □

The important intuition is that such a "modulo α intersection" can always be represented by at most two intervals. Remark that three last sets are the only ones which are represented by $2 > 1$ intervals.

While Lemma 43 gives structure to intersections of two modular projections of intervals, the next lemma reveals how many intervals will be required to represent a *one-to-many* intersection. We will use this bound in every step of our algorithm. We want to add that both of these lemmas and even Lemma 45 do not depend on the harmonic property by themselves. However, they turn out to be especially useful in this setting.

Lemma 44. *Let $\alpha \ge 1$, let v be an interval and let Q be a set of $k \ge 1$ intervals. Then there is a set R of at most $k+1$ intervals s.t. $v^{[\alpha]} \cap (\bigcup Q)^{[\alpha]} = (\bigcup R)^{[\alpha]}$.*

Proof. We simply obtain that

$$v^{[\alpha]} \cap \Big(\bigcup Q\Big)^{[\alpha]} = \bigcup_{w \in Q} (v^{[\alpha]} \cap w^{[\alpha]}) = \bigcup_{\substack{w \in Q \\ w^{[\alpha]} \subseteq v^{[\alpha]}}} w^{[\alpha]} \cup \bigcup_{w \in D} (v^{[\alpha]} \cap w^{[\alpha]})$$

where $D = \{ w \in Q \mid w^{[\alpha]} \not\subseteq v^{[\alpha]}, w^{[\alpha]} \cap v^{[\alpha]} \neq \emptyset \}$ denotes the subset of intervals that cause the interesting intersections with $v^{[\alpha]}$ (cf. Lemma 43). Obviously, all other intersections can be represented by at most one interval each. So we study the intersections with D. In fact, everything gets simple if there are $w_1, w_2 \in D$ such that $v^{[\alpha]} \cap w_1^{[\alpha]} = \varphi_\alpha(v, w_1) \dot\cup \psi_\alpha(v, w_1)^{[\alpha]}$ and $v^{[\alpha]} \cap w_2^{[\alpha]} = \varphi_\alpha(w_2, v) \dot\cup \psi_\alpha(v, w_2)^{[\alpha]}$. By simply adapting the inequalities of the first case distinction in the proof of Lemma 43 we find

$$\begin{aligned}
&(v^{[\alpha]} \cap w_1^{[\alpha]}) \cup (v^{[\alpha]} \cap w_2^{[\alpha]}) \\
&= ([0, u_v^{[\alpha]}] \dot\cup [\ell_v^{[\alpha]}, u_{w_1}^{[\alpha]}] \dot\cup [\ell_{w_1}^{[\alpha]}, \alpha)) \\
&\qquad \cup ([0, u_{w_2}^{[\alpha]}] \dot\cup [\ell_{w_2}^{[\alpha]}, u_v^{[\alpha]}] \dot\cup [\ell_v^{[\alpha]}, \alpha)) \\
&= [0, u_v^{[\alpha]}] \dot\cup [\ell_v^{[\alpha]}, \alpha) = v^{[\alpha]}
\end{aligned}$$

which implies that $v^{[\alpha]} \cap (\bigcup Q)^{[\alpha]} = v^{[\alpha]}$ can be represented by only one interval, namely v. Therefore, in order to get an upper bound we assume that these two types of intersections do not come together. In more detail, we may assume by symmetry that $D = D_1 \dot\cup D_2$ where

$$\begin{aligned}
D_1 &= \{ w \in D \mid v^{[\alpha]} \cap w^{[\alpha]} = \varphi_\alpha(v,w) \dot\cup \varphi_\alpha(w,v) \} \text{ and} \\
D_2 &= \{ w \in D \mid v^{[\alpha]} \cap w^{[\alpha]} = \varphi_\alpha(v,w) \dot\cup \psi_\alpha(v,w)^{[\alpha]} \}.
\end{aligned}$$

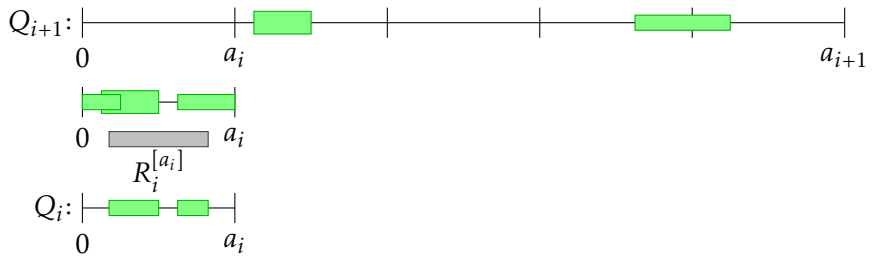


Figure 6.4: A step from $i+1$ to i; modular projection to $[0, a_i)$ and intersection with $R_i^{[a_i]}$

It turns out that

$$\begin{aligned}\bigcup_{w\in D_1}(v^{[\alpha]}\cap w^{[\alpha]}) &= \bigcup_{w\in D_1}([\ell_v^{[\alpha]}, u_w^{[\alpha]}]\,\dot\cup\,[\ell_w^{[\alpha]}, u_v^{[\alpha]}])\\ &= [\ell_v^{[\alpha]}, \max_{w\in D_1} u_w^{[\alpha]}]\cup[\min_{w\in D_1}\ell_w^{[\alpha]}, u_v^{[\alpha]}] \quad \text{and}\\ \bigcup_{w\in D_2}(v^{[\alpha]}\cap w^{[\alpha]}) &= \bigcup_{w\in D_2}([\ell_v^{[\alpha]}, u_w^{[\alpha]}]\,\dot\cup\,[\ell_w^{[\alpha]}, \alpha+u_v^{[\alpha]}]^{[\alpha]})\\ &= [\ell_v^{[\alpha]}, \max_{w\in D_2} u_w^{[\alpha]}]\cup[\min_{w\in D_2}\ell_w^{[\alpha]}, \alpha+u_v^{[\alpha]}]^{[\alpha]}\end{aligned}$$

which finally joins up to

$$\bigcup_{w\in D}(v^{[\alpha]}\cap w^{[\alpha]}) = [\ell_v^{[\alpha]}, \max_{w\in D} u_w^{[\alpha]}]\cup[\min_{w\in D}\ell_w^{[\alpha]}, \alpha+u_v^{[\alpha]}]^{[\alpha]}.$$

Hence, all intersections with intervals in D may be represented by at most two intervals in total while each other intersection can be represented by at most one interval. Thus, if $|D| = 0$ then the whole intersection can be represented by at most k intervals. If $|D| \geq 1$, then there are at most $2 + |Q| - |D| \leq 2 + k - 1 = k + 1$ intervals required. □

Algorithm 9 Feasibility test for FSC

procedure FEASIBLE($I = (a_1, \ldots, a_n, R_1, \ldots, R_n)$)
  $Q_n \leftarrow \{R_n\}$
  for $i = n-1, \ldots, 1$ **do**
    Compute set Q_i such that
      $(\bigcup Q_i)^{[a_i]} = R_i^{[a_i]} \cap (\bigcup Q_{i+1})^{[a_i]}$ and $|Q_i| \leq \mathcal{O}(n-i)$
  if $\bigcup Q_1 = \varnothing$ **then**
    return NO
  else
    return YES

Let S_i denote the set of all solutions $s \in \mathbb{Z}_{\geq 0}$ that are feasible for each of the constraints $i, i+1, \ldots, n$. We set $S_{n+1} = \mathbb{Z}_{\geq 0}$ to denote the feasible solutions to an empty set of constraints. The correctness of Algorithm 9 is

implied by the following fundamental Lemma. See Figure 6.4 for an example of a step inside the algorithm.

Lemma 45. *It holds true that* $S_i^{[a_i]} = R_i^{[a_i]} \cap S_{i+1}^{[a_i]}$ *for all* $i \in [n]$.

Proof. Let $r \in S_i^{[a_i]}$. So there is a solution $s \in S_i$ such that $r = s^{[a_i]} \in R_i^{[a_i]}$. It holds that $S_i \subseteq S_{i+1}$ which implies $s \in S_{i+1}$ and thus $r = s^{[a_i]} \in S_{i+1}^{[a_i]}$.

Vice-versa let $r \in R_i^{[a_i]} \cap S_{i+1}^{[a_i]}$. So there is a solution $s \in S_{i+1}$ with $s^{[a_i]} = r$. From $r \in R_i^{[a_i]}$ we get $s^{[a_i]} \in R_i^{[a_i]}$. Hence, $s \in S_i$ and $r = s^{[a_i]} \in S_i^{[a_i]}$. □

Theorem 27. *Algorithm 9 decides the feasibility of an instance in time* $\mathcal{O}(n^2)$.

Proof. We show that $\bigcup Q_i \equiv S_i \pmod{a_i}$ for all $i = n, \dots, 1$. This will prove the algorithm correct since then $\bigcup Q_1 \equiv S_1 \pmod{a_1}$ and that means $\bigcup Q_1$ is empty if and only if S_1 is empty. Obviously it holds that $\bigcup Q_n \equiv S_n \pmod{a_n}$ since $\bigcup Q_n = R_n$. Now suppose that $\bigcup Q_{i+1} \equiv S_{i+1} \pmod{a_{i+1}}$ for some $i \geq 1$. We have that $(\bigcup Q_i)^{[a_i]} = R_i^{[a_i]} \cap (\bigcup Q_{i+1})^{[a_i]}$ where the harmonic property implies $(\bigcup Q_{i+1})^{[a_i]} = ((\bigcup Q_{i+1})^{[a_{i+1}]})^{[a_i]} = (S_{i+1}^{[a_{i+1}]})^{[a_i]} = S_{i+1}^{[a_i]}$. Together with Lemma 45 this yields $(\bigcup Q_i)^{[a_i]} = R_i^{[a_i]} \cap S_{i+1}^{[a_i]} = S_i^{[a_i]}$ and that proves the algorithm correct. Using Lemmas 43 to 45 each set Q_i can be computed in time $\mathcal{O}(n)$ and this yields a total running time of $\mathcal{O}(n^2)$. □

6.3.2 *Optimal solutions*

Unfortunately, Algorithm 9 neither calculates a solution nor it directly implies one. Here we present an algorithm to compute the smallest feasible solution $s_{\min}$ to FSC. However, by searching in the opposite direction the same technique also applies to the computation of the largest feasible solution $s_{\max} < a_n$. We start with a simple binary search approach.

Corollary 4. *For feasible instances* $s_{\min}$ *is computable in time* $\mathcal{O}(n^2 \log(a_n))$.

This can be achieved by introducing an additional constraint measuring the value of s as follows. Let β be a positive integer. We extend the problem instance by a new constraint with number $n+1$ defined by $a_{n+1} = 2 \cdot a_n$, $b_{n+1} = 0$, and $B_{n+1} = \beta$. Remark that this *β-instance* admits the same set of solutions as the original instance as long as β is large enough, e. g. $\beta = a_n$ (cf. Observation 16). Consider a feasible solution to the β-instance where $\beta \leq a_n$. It holds that

$$2a_n x_{n+1} = a_{n+1} x_{n+1} \leq s + a_{n+1} x_{n+1} \leq B_{n+1} = \beta \leq a_n$$

which implies $x_{n+1} \leq \lfloor \frac{1}{2} \rfloor = 0$. However, if $x_{n+1} < 0$ then $s \geq a_{n+1} \cdot |x_{n+1}|$ and therefore the solution $s' = s + a_{n+1} x_{n+1}$ with $x'_{n+1} = 0$ and $x'_i = x_i - (a_{n+1}/a_i) x_{n+1}$ for all $i = 1, \dots, n$ is better than s and $x'_{n+1} = 0$.

Thus we may assume generally that $x_{n+1} = 0$ which allows us to measure the value of s using the upper bound β. We use β to do a binary search in the interval $[0, a_n]$ using Algorithm 9 to check the β-instance for feasibility. The smallest possible value for β then states the optimum value and that proves

Corollary 4. However, with additional ideas we are able to achieve strongly polynomial time. We want to give some helpful intuition first.

Clearly, after revealing the intervals in Q_1 with Algorithm 9 a straightforward idea is to try tracing them back to a small solution for s, but routing through the modulus operations appears to become a non-polynomial bottleneck.

However, the following idea is a first step to end up with a constraint aggregation approach. Given the projections $A^{[ab]}$ and $B^{[a]}$ of two sets $A, B \subseteq \mathbb{Z}$ one can compute the intersection $A^{[a]} \cap B^{[a]}$ in at least two ways; primitively we compute $(A^{[ab]})^{[a]} = A^{[a]}$ and then intersect it with $B^{[a]}$, but also we can intersect $A^{[ab]}$ with b translated copies $B^{[a]}, a + B^{[a]}, \dots, (b-1)a + B^{[a]}$ of $B^{[a]}$ before computing the $[a]$-projection. In fact, the following lemma seems to be a characteristic property of modular arithmetic on sets.

Lemma 46. *For all numbers $a, b \in \mathbb{Z}_{\geq 1}$ and sets $A, B \subseteq \mathbb{Z}$ it holds*

$$A^{[a]} \cap B^{[a]} = \left(A^{[ab]} \cap \bigcup_{i=0}^{b-1} (ia + B^{[a]}) \right)^{[a]}.$$

Proof. Let x be a number. Then it holds

$$\begin{aligned} & x \in \left(A^{[ab]} \cap \bigcup_{i=0}^{b-1} (ia + B^{[a]}) \right)^{[a]} \\ \Leftrightarrow \;& \exists y \in A^{[ab]} : y \in \bigcup_{i=0}^{b-1} (ia + B^{[a]}) \wedge x = y^{[a]} \\ \Leftrightarrow \;& \exists y \in A^{[ab]} : y^{[a]} \in B^{[a]} \wedge x = y^{[a]} \\ \Leftrightarrow \;& x \in A^{[a]} \cap B^{[a]} \end{aligned}$$

where the last equivalence follows from $(A^{[ab]})^{[a]} = A^{[a]}$. □

Since the right side can be written as the modular projection of a *union of intersections* we can find a sensible strengthening; in fact, for arbitrary sets $X, M_0, \dots, M_{m-1}$ it holds that

$$\bigcup_{i=0}^{m-1} (X \cap M_i) = \bigcup_{i=0}^{m-1} (X \cap (M_i \setminus \bigcup_{j=0}^{i-1} (X \cap M_j))).$$

While the left-hand side may not, the right-hand side is always a *disjoint* union. Taking into account the modular projections this leads to the following corollary.

Corollary 5. *For all numbers $a, b \in \mathbb{Z}_{\geq 1}$ and sets $A, B \subseteq \mathbb{Z}$ it holds $A^{[a]} \cap B^{[a]} = (\bigcup_{i=0}^{b-1} D_i)^{[a]}$ where $D_i = A^{[ab]} \cap Y_i$ and $Y_i = ia + (B^{[a]} \setminus \bigcup_{j=0}^{i-1} D_j^{[a]})$ for all $i = 0, \dots, b-1$.*

We will use Corollary 5 to aggregate constraints in order to reduce the problem size. The following observation gives a first bound for the smallest feasible solution $s_{\min}$.

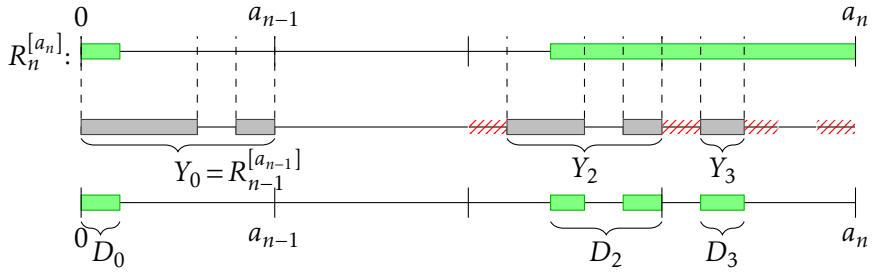


Figure 6.5: An example of four required intervals to represent $R_{n-1}^{[a_{n-1}]} \cap R_n^{[a_{n-1}]}$ in Lemma 47

Observation 17. *For feasible instances it holds that* $s_{\min} \in R_n^{[a_n]}$.

This is true since in the harmonic case $s_{\min} < \operatorname{lcm}(a_1, \ldots, a_n) = a_n$ due to Observation 16 which then implies that $s_{\min} = s_{\min}^{[a_n]} \in R_n^{[a_n]}$ using Observation 15. Motivated by Observation 17 the idea is to search for $s_{\min}$ in the modular projection $R_n^{[a_n]}$ by aggregating the penultimate constraint $n-1$ into the last constraint n. In fact, the number of required intervals to represent both constraints can be bounded by a constant. A fine-grained construction then enforces the algorithm to efficiently iterate the feasibility test on aggregated instances to find the optimum value.

Theorem 28. *For feasible instances* $s_{\min}$ *can be computed in time* $\mathcal{O}(n^3)$.

Remark that the set of feasible solutions for the last two constraints is $S_{n-1} = R_{n-1}^{[a_{n-1}]} \cap (R_n^{[a_n]})^{[a_{n-1}]} = R_{n-1}^{[a_{n-1}]} \cap R_n^{[a_{n-1}]}$. Therefore, the next lemma states the crucial argument of the algorithm.

Lemma 47. *The intersection* $R_{n-1}^{[a_{n-1}]} \cap R_n^{[a_{n-1}]}$ *can always be represented by the disjoint union* $U \subseteq R_n^{[a_n]}$ *of only constant many intervals in* $R_n^{[a_n]}$ *such that*

(a) $U^{[a_{n-1}]} = R_{n-1}^{[a_{n-1}]} \cap R_n^{[a_{n-1}]}$ *and*

(b) $u \equiv r \pmod{a_{n-1}}$ *implies* $u \leq r$ *for all* $u \in U$, $r \in R_n^{[a_n]}$.

Here the former property states that indeed the intervals in U are a proper representation for the last two constraints. The important property is the latter; in fact, it ensures that U is the best possible representation in the sense that U consists of the *smallest* intervals possible (see Figure 6.5).

Proof of Lemma 47. (a). By defining $D_i = Y_i \cap R_n^{[a_n]}$ and

$$Y_i = i a_{n-1} + (R_{n-1}^{[a_{n-1}]} \setminus \bigcup_{j=0}^{i-1} D_j^{[a_{n-1}]})$$

for all $i \in \{0, \ldots, a_n/a_{n-1} - 1\}$ Corollary 5 proves the claim (cf. Figure 6.5). (b) follows by construction.

It remains to show that $\bigcup_i D_i$ is the union of only constant many disjoint intervals. Apparently, the intervals are disjoint by construction.

We claim that there are at most three non-empty sets D_i. Assume there are at least four non-empty translates D_i, namely D_i, D_j, D_k, D_ℓ. Then, since R_n is an interval it holds for at least two $p, q \in \{i, j, k, \ell\}$ that the *full interval translates* $F_p = [pa_{n-1}, (p+1)a_{n-1})$ and $F_q = [qa_{n-1}, (q+1)a_{n-1})$ are subsets of $R_n^{[a_n]}$. For p (and also for q) we get

$$D_p^{[a_{n-1}]} = (\underbrace{Y_p}_{\subseteq F_p} \cap R_n^{[a_n]})^{[a_{n-1}]} = Y_p^{[a_{n-1}]} = R_{n-1}^{[a_{n-1}]} \setminus \bigcup_{j=0}^{p-1} D_j^{[a_{n-1}]}$$

which implies with $\bigcup_{j=0}^{p-1} D_j^{[a_{n-1}]} \subseteq R_{n-1}^{[a_{n-1}]}$ that

$$\bigcup_{j=0}^{p} D_j^{[a_{n-1}]} = D_p^{[a_{n-1}]} \cup \bigcup_{j=0}^{p-1} D_j^{[a_{n-1}]} = R_{n-1}^{[a_{n-1}]}.$$

Then it follows $\bigcup_{j=0}^{p} D_j^{[a_{n-1}]} = R_{n-1}^{[a_{n-1}]} = \bigcup_{j=0}^{q} D_j^{[a_{n-1}]}$. W.l.o.g. let $p < q$. Then $D_q = Y_q \cap R_n^{[a_n]}$ is empty since

$$Y_q = qa_{n-1} + \left(R_{n-1}^{[a_{n-1}]} \setminus \bigcup_{j=0}^{q-1} D_j^{[a_{n-1}]} \right) \subseteq qa_{n-1} + \left(R_{n-1}^{[a_{n-1}]} \setminus R_{n-1}^{[a_{n-1}]} \right)$$

is empty and we have a contradiction.

Using the same case distinctions as in the proof of Lemma 43 one can show that each set D_i consists of at most two intervals. Therefore, all the non-empty sets D_i consist of at most $3 \cdot 2 = 6$ intervals in total. In fact, one can improve this bound to a total number of at most 4 intervals (see Figure 6.5) by a more sophisticated case distinction. □

This admits an algorithm using an aggregation argument as follows. For constraints n and $n-1$ we use Lemma 47 to compute disjoint intervals $E_1, \ldots, E_k \subseteq R_n^{[a_n]}$ (representing the constraints n and $n-1$) where $k \leq C$ for a small constant C. If $k \geq 1$ then use Algorithm 9 to check the feasibility of the instances $\mathcal{I}_1, \ldots, \mathcal{I}_k$ defined by

$$(\mathcal{I}_j) \qquad \min\{ s \mid s^{[a_i]} \in R_i^{[a_i]} \forall i = 1, \ldots, n-2,\ s^{[a_n]} \in E_j^{[a_n]}, s \in \mathbb{Z}_{\geq 0} \}.$$

If none of the instances $\mathcal{I}_1, \ldots, \mathcal{I}_k$ admits a solution then the original instance can not be feasible. Assume that there is at least one feasible instance. Now, since $E_1, \ldots, E_k$ are disjoint exactly one of them contains the optimum value for s. W.l.o.g. assume that $E_1 < \cdots < E_k$. Then there is a smallest index j such that $\mathcal{I}_j$ is feasible and we solve $\mathcal{I}_j$ recursively to find the optimum value. Together this yields an algorithm running in time $n \cdot C \cdot \mathcal{O}(n^2) = \mathcal{O}(n^3)$.

6.4 HARDNESS OF BMS

We reduce from DIRECTED DIOPHANTINE APPROXIMATION with *rounding down*. For any vector $v \in \mathbb{R}^n$ let $\lfloor v \rfloor$ denote the vector where each component is rounded down, i. e. $(\lfloor v \rfloor)_i = \lfloor v_i \rfloor$ for all $i \leq n$.

DIRECTED DIOPHANTINE APPROXIMATION with rounding down (DDA$^{\downarrow}$)
Given: $\alpha_1, \ldots, \alpha_n \in \mathbb{Q}_+$, $N \in \mathbb{Z}_{\geq 1}$, $\varepsilon \in \mathbb{Q}$, $0 < \varepsilon < 1$
Decide whether there is a $Q \in [N]$ such that $\|Q\alpha - \lfloor Q\alpha \rfloor\|_\infty \leq \varepsilon$.

Eisenbrand and Rothvoß proved that DDA$^{\downarrow}$ is NP-hard [46]. In fact, every instance of DDA$^{\downarrow}$ can be expressed as a BMS instance, which yields the following theorem.

Theorem 29. BMS *is* NP*-hard (even if* $b_i = 0$ *for all* i *with* $a_i \neq 0$*).*

Proof. Write $\alpha_i = \beta_i/\gamma_i$ for integers $\beta_i \geq 0, \gamma_i \geq 1$ and set $\lambda = \prod_j \beta_j$. Then $\lambda/\alpha_i = (\lambda/\beta_i)\gamma_i \geq 0$ is integer. Let $\mathcal{M}$ denote the following instance of BMS:

$$
\begin{aligned}
0 \;\leq\; Q' - (\lambda/\alpha_i)\cdot y_i \quad &\leq \quad \lfloor (\lambda/\alpha_i)\cdot \varepsilon \rfloor \qquad \forall i = 1,\ldots,n && (6.1)\\
\lambda \;\leq\; Q' - 0 \cdot y_{n+1} \quad &\leq \quad \lambda \cdot N && (6.2)\\
0 \;\leq\; Q' - \lambda \cdot y_{n+2} \quad &\leq \quad 0 && (6.3)\\
& \quad Q', y_i \in \mathbb{Z} \qquad \forall i = 1,\ldots,n+2
\end{aligned}
$$

So let $Q \in \{1,\ldots,N\}$ with $\|Q\alpha - \lfloor Q\alpha \rfloor\|_\infty \leq \varepsilon$ be given. We obtain readily that $Q' = \lambda Q$ and $y = (\lfloor Q\alpha_1 \rfloor, \ldots, \lfloor Q\alpha_n \rfloor, 0, Q)$ defines a solution of $\mathcal{M}$ since

$$0 \leq Q\alpha_i - \lfloor Q\alpha_i \rfloor \leq \varepsilon$$

if and only if

$$0 \leq \underbrace{\lambda Q - (\lambda/\alpha_i)\cdot \lfloor Q\alpha_i \rfloor}_{\in \mathbb{Z}} \leq (\lambda/\alpha_i)\cdot\varepsilon.$$

Vice-versa let (Q', y) be a solution to $\mathcal{M}$. We see that (6.1) implies that

$$0 \leq Q' - (\lambda/\alpha_i)\cdot y_i \leq \lfloor (\lambda/\alpha_i)\cdot\varepsilon \rfloor \leq (\lambda/\alpha_i)\cdot\varepsilon$$

and by (6.3) we get $Q' = \lambda \cdot y_{n+2}$ which then implies $0 \leq y_{n+2}\alpha_i - y_i \leq \varepsilon < 1$ for all $i \leq n$. Now, since y_i is integer, there can be only one value for y_i, i. e. $y_i = \lfloor y_{n+2}\alpha_i \rfloor$. By $Q' = \lambda \cdot y_{n+2}$ and (6.2) we get $y_{n+2} \in \{1,\ldots,N\}$ and by setting $Q = y_{n+2}$ this yields $\|Q\alpha - \lfloor Q\alpha \rfloor\|_\infty \leq \varepsilon$ and that proves the claim. □

6.5 OPEN QUESTIONS

So far, we are able to optimize the simple objective function $f(s,x) = s$. The question arises, whether the optimization of a general linear function f (which also depends on x) can be done in polynomial time as well.

Furthermore, it remains interesting to search for an even faster algorithm to decide the feasibility of an instance in sub-quadratic time which would

lead to an algorithm to find $s_{\min}$ in sub-cubic time. Also, it remains open whether the feasibility testing can be bypassed efficiently.

Finally, the Chinese Remainder Theorem especially solves the case that all divisors are pairwise coprime. Is there a corresponding result for FSC as well?

BIBLIOGRAPHY

Never memorize something that you can look up.

— Albert Einstein

[6] Manindra Agrawal and Somenath Biswas. Primality and identity testing via chinese remaindering. *J. ACM*, 50(4):429–443, 2003.

[7] Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling. In Michael E. Saks, editor, *Proc. SODA 1997*, pages 493–500. ACM/SIAM, 1997.

[8] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. Hard real-time scheduling: The deadline-monotonic approach. *IFAC Proceedings Volumes*, 24(2):127–132, 1991. IFAC/IFIP Workshop on Real Time Programming, Atlanta, GA, USA, 15-17 May 1991.

[9] Neil C. Audsley, Alan Burns, Mike M. Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Softw. Eng. J.*, 8(5):284–292, 1993.

[10] Brenda S. Baker and Edward G. Coffman Jr. Mutual exclusion scheduling. *Theor. Comput. Sci.*, 162(2):225–243, 1996.

[11] Sanjoy K. Baruah. Efficient computation of response time bounds for preemptive uniprocessor deadline monotonic scheduling. *Real Time Syst.*, 47(6):517–533, 2011.

[12] Sanjoy K. Baruah. Work in progress: The ilp-tractability of schedulability analysis problems. In *Proc. RTSS 2020*, pages 391–394. IEEE, 2020.

[13] Sanjoy K. Baruah and Joël Goossens. Scheduling real-time tasks. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[14] Sanjoy K. Baruah, Louis E. Rosier, and Rodney R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real Time Syst.*, 2(4):301–324, 1990.

[15] József Beck and Tibor Fiala. "Integer-making" theorems. *Discrete Applied Mathematics*, 3(1):1–8, 1981.

[16] Debe Bednarchak and Martin Helm. A note on the beck-fiala theorem. *Comb.*, 17(1):147–149, 1997.

[17] Enrico Bini, Thi Huyen Chau Nguyen, Pascal Richard, and Sanjoy K. Baruah. A response-time bound in fixed-priority scheduling with arbitrary deadlines. *IEEE Trans. Computers*, 58(2):279–286, 2009.

[18] Jacek Blazewicz, Nadia Brauner, and Gerd Finke. Scheduling with discrete resource constraints. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[19] Jacek Blazewicz, Klaus H. Ecker, Erwin Pesch, Günter Schmidt, Malgorzata Sterna, and Jan Weglarz. *Scheduling under Resource Constraints*, pages 475–525. Springer International Publishing, Cham, 2019.

[20] Jacek Blazewicz, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discret. Appl. Math.*, 5(1):11–24, 1983.

[21] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.

[22] Hans L. Bodlaender and Klaus Jansen. On the complexity of scheduling incompatible jobs with unit-times. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *Proc. MFCS 1993*, volume 711 of *Lecture Notes in Computer Science*, pages 291–300. Springer, 1993.

[23] Hans L. Bodlaender, Klaus Jansen, and Gerhard J. Woeginger. Scheduling with incompatible jobs. *Discret. Appl. Math.*, 55(3):219–232, 1994.

[24] Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Nicole Megow, and Andreas Wiese. Polynomial-time exact schedulability tests for harmonic real-time tasks. In *Proc. RTSS 2013*, pages 236–245. IEEE Computer Society, 2013.

[25] Édouard Bonnet and Florian Sikora. The PACE 2018 parameterized algorithms and computational experiments challenge: The third iteration. In Christophe Paul and Michal Pilipczuk, editors, *Proc. IPEC 2018*, volume 115 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[26] Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theor. Comput. Sci.*, 276(1-2):17–32, 2002.

[27] Bo Chen. A better heuristic for preemptive parallel machine scheduling with batch setup times. *SIAM J. Comput.*, 22(6):1303–1318, 1993.

[28] Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *J. Comput. Syst. Sci.*, 96:1–32, 2018.

[29] Lin Chen, Martin Koutecký, Lei Xu, and Weidong Shi. New bounds on augmenting steps of block-structured integer programs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *Proc. ESA 2020*, volume 173 of *LIPIcs*, pages 33:1–33:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[30] Michele Conforti, Marco Di Summa, and Laurence A. Wolsey. The mixing set with divisible capacities. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Proc. IPCO 2008*, volume 5035 of *Lecture Notes in Computer Science*, pages 435–449. Springer, 2008.

[31] Michele Conforti and Giacomo Zambelli. The mixing set with divisible capacities: A simple approach. *Oper. Res. Lett.*, 37(6):379–383, 2009.

[32] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proc. STOC 1971*, pages 151–158. ACM, 1971.

[33] José R. Correa, Alberto Marchetti-Spaccamela, Jannik Matuschke, Leen Stougie, Ola Svensson, Víctor Verdugo, and José Verschae. Strong LP formulations for scheduling splittable jobs on unrelated machines. *Math. Program.*, 154(1-2):305–328, 2015.

[34] Jana Cslovjecsek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-structured integer and linear programming in strongly polynomial and near linear time. In Dániel Marx, editor, *Proc. SODA 2021*, pages 1666–1681. SIAM, 2021.

[35] Jana Cslovjecsek, Friedrich Eisenbrand, Michal Pilipczuk, Moritz Venzin, and Robert Weismantel. Efficient sequential and parallel algorithms for multistage stochastic integer programming using proximity. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *Proc. ESA 2021*, volume 204 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[36] Syamantak Das and Andreas Wiese. On minimizing the makespan when some jobs cannot be assigned on the same machine. In Kirk Pruhs and Christian Sohler, editors, *Proc. ESA 2017*, volume 87 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[37] Robert I. Davis and Alan Burns. Response time upper bounds for fixed priority real-time systems. In *Proc. RTSS 2008*, pages 407–418. IEEE Computer Society, 2008.

[38] Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In Daniel Lokshtanov and Naomi Nishimura, editors, *Proc. IPEC 2017*, volume 89 of *LIPIcs*, pages 30:1–30:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[39] Max A. Deppert. Approximation algorithms for scheduling problems with batch setup times. Master's thesis, Kiel University, March 2018.

[40] György Dósa, Hans Kellerer, and Zsolt Tuza. Restricted assignment scheduling with resource constraints. *Theor. Comput. Sci.*, 760:72–87, 2019.

[41] M. Ayaz Dzulfikar, Johannes Klaus Fichte, and Markus Hecher. The PACE 2019 parameterized algorithms and computational experiments challenge: The fourth iteration (invited paper). In Bart M. P. Jansen and Jan Arne Telle, editors, *Proc. IPEC 2019*, volume 148 of *LIPIcs*, pages 25:1–25:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[42] Emrah B. Edis, Ceyda Oguz, and Irem Ozkarahan. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *Eur. J. Oper. Res.*, 230(3):449–463, 2013.

[43] Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *Proc. ICALP 2018*, volume 107 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[44] Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *CoRR*, abs/1904.01361, 2019.

[45] Friedrich Eisenbrand and Thomas Rothvoß. Static-priority real-time scheduling: Response time computation is np-hard. In *Proc. RTSS 2008*, pages 397–406. IEEE Computer Society, 2008.

[46] Friedrich Eisenbrand and Thomas Rothvoß. New hardness results for diophantine approximation. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Proc. APPROX 2009*, volume 5687 of *Lecture Notes in Computer Science*, pages 98–110. Springer, 2009.

[47] Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020.

[48] Pontus Ekberg. Rate-monotonic schedulability of implicit-deadline tasks is np-hard beyond liu and layland's bound. In *Proc. RTSS 2020*, pages 308–318. IEEE, 2020.

[49] Guy Even, Magnús M. Halldórsson, Lotem Kaplan, and Dana Ron. Scheduling with conflicts: online and offline algorithms. *J. Sched.*, 12(2):199–224, 2009.

[50] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proc. IEEE*, 93(2):216–231, 2005.

[51] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[52] Oded Goldreich, Dana Ron, and Madhu Sudan. Chinese remaindering with errors. *IEEE Trans. Inf. Theory*, 46(4):1330–1338, 2000.

[53] Kilian Grage, Klaus Jansen, and Kim-Manuel Klein. An EPTAS for machine scheduling with bag-constraints. In Christian Scheideler and Petra Berenbrink, editors, *Proc. SPAA 2019*, pages 135–144. ACM, 2019.

[54] Ron Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1977.

[55] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[56] Alexander Grigoriev and Marc Uetz. Scheduling jobs with time-resource tradeoff via nonlinear programming. *Discret. Optim.*, 6(4):414–419, 2009.

[57] Oktay Günlük and Yves Pochet. Mixing mixed-integer inequalities. *Math. Program.*, 90(3):429–457, 2001.

[58] J. N. D. Gupta and A. J. Ruiz-Torres. A listfit heuristic for minimizing makespan on identical parallel machines. *Production Planning & Control*, 12(1):28–36, 2001.

[59] Jatinder ND Gupta and Alex J Ruiz-Torres. A listfit heuristic for minimizing makespan on identical parallel machines. *Production Planning & Control*, 12(1):28–36, 2001.

[60] Venkatesan Guruswami, Amit Sahai, and Madhu Sudan. "soft-decision" decoding of chinese remainder codes. In *Proc. FOCS 2000*, pages 159–168. IEEE Computer Society, 2000.

[61] Emmanuel Hebrard, Marie-José Huguet, Nicolas Jozefowiez, Adrien Maillard, Cédric Pralet, and Gérard Verfaillie. Approximation of the parallel machine scheduling problem with additional unit resources. *Discret. Appl. Math.*, 215:126–135, 2016.

[62] Raymond Hemmecke, Matthias Köppe, and Robert Weismantel. A polynomial-time algorithm for optimizing over N-fold 4-block decomposable integer programs. In Friedrich Eisenbrand and F. Bruce Shepherd, editors, *Proc. IPCO 2010*, volume 6080 of *Lecture Notes in Computer Science*, pages 219–229. Springer, 2010.

[63] Raymond Hemmecke and Rüdiger Schultz. Decomposition of test sets in stochastic integer programming. *Math. Program.*, 94(2-3):323–341, 2003.

[64] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987.

[65] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23(2):317–327, 1976.

[66] Clay Mathematics Institute. P vs NP Problem. https://claymath.org/millennium-problems/p-vs-np-problem. Accessed: 2022-08-01.

[67] Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-ip: new PTAS results for scheduling with setup times. *Math. Program.*, 195(1):367–401, 2022.

[68] Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Math. Oper. Res.*, 45(4):1371–1392, 2020.

[69] Klaus Jansen and Felix Land. Non-preemptive scheduling with setup times: A PTAS. In Pierre-François Dutot and Denis Trystram, editors, *Proc. EUROPAR 2016*, volume 9833 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2016.

[70] Klaus Jansen, Alexandra Lassota, and Marten Maack. Approximation algorithms for scheduling with class constraints. In Christian Scheideler and Michael Spear, editors, *Proc. SPAA 2020*, pages 349–357. ACM, 2020.

[71] Klaus Jansen, Marten Maack, and Alexander Mäcker. Scheduling on (un-)related machines with setup times. In *Proc. IPDPS 2019*, pages 145–154. IEEE, 2019.

[72] Klaus Jansen, Marten Maack, and Malin Rau. Approximation schemes for machine scheduling with resource (in-)dependent processing times. *ACM Trans. Algorithms*, 15(3):31:1–31:28, 2019.

[73] Klaus Jansen and Malin Rau. Closing the gap for single resource constraint scheduling. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *Proc. ESA 2021*, volume 204 of *LIPIcs*, pages 53:1–53:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[74] Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In Avrim Blum, editor, *Proc. ITCS 2019*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[75] Klaus Jansen and Lars Rohwedder. On integer programming, discrepancy, and convolution, 2019. Extension of [74] by the theory of discrepancy.

[76] Klaus Jansen and Lars Rohwedder. On integer programming, discrepancy, and convolution. *Mathematics of Operations Research*, 2022.

[77] Teun Janssen. *Optimization in the Photolithography Bay: Scheduling and the Traveling Salesman Problem.* PhD thesis, Delft University of Technology, Netherlands, 2019.

[78] Teun Janssen, Céline M. F. Swennenhuis, Abdoul Bitar, Thomas Bosman, Dion Gijswijt, Leo van Iersel, Stéphane Dauzère-Pérès, and Claude Yugma. Parallel machine scheduling with a single resource per job. *CoRR*, abs/1809.05009, 2018.

[79] Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

[80] Edward G. Coffman Jr., M. R. Garey, and David S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.*, 7(1):1–17, 1978.

[81] Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.

[82] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

[83] SK Kedia. A job scheduling problem with parallel processors. *Unpublished report, Department of Industrial and Operations Engineering. University of Michigan, Ann Arbor, MI*, 1971.

[84] Hans Kellerer and Vitaly A. Strusevich. Scheduling problems for parallel dedicated machines under multiple resource constraints. *Discret. Appl. Math.*, 133(1-3):45–68, 2003.

[85] Kim-Manuel Klein. About the complexity of two-stage stochastic ips. *Math. Program.*, 192(1):319–337, 2022.

[86] Peter Kling, Alexander Mäcker, Sören Riechers, and Alexander Skopalik. Sharing is caring: Multiprocessor scheduling with a sharable resource. In Christian Scheideler and Mohammad Taghi Hajiaghayi, editors, *Proc. SPAA 2017*, pages 123–132. ACM, 2017.

[87] Donald Ervin Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition.* Addison-Wesley, 1998.

[88] Abey Kuruvilla and Giuseppe Paletta. Minimizing makespan on identical parallel machines. *Int. J. Oper. Res. Inf. Syst.*, 6(1):19–29, 2015.

[89] Chung-Yee Lee and J. David Massey. Multiprocessor scheduling: combining LPT and MULTIFIT. *Discret. Appl. Math.*, 20(3):233–242, 1988.

[90] John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proc. RTSS 1990*, pages 201–209. IEEE Computer Society, 1990.

[91] Joseph Y.-T. Leung. Bin packing with restricted piece sizes. *Inf. Process. Lett.*, 31(3):145–149, 1989.

[92] Joseph Y.-T. Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform. Evaluation*, 2(4):237–250, 1982.

[93] L. A. Levin. Universal enumeration problems. *Problemy Peredači Informacii*, 9(3):115–116, 1973. (in Russian).

[94] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[95] Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. Non-preemptive scheduling on machines with setup times. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Proc. WADS 2015*, volume 9214 of *Lecture Notes in Computer Science*, pages 542–553. Springer, 2015.

[96] Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.

[97] Robert McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6(1):1–12, 1959.

[98] Andrew J. Miller and Laurence A. Wolsey. Tight formulations for some simple mixed integer programs and convex objective integer programs. *Math. Program.*, 98(1-3):73–88, 2003.

[99] Aloysius Ka-Lau Mok. *Fundamental design problems of distributed systems for the hard-real-time environment.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

[100] Clyde L. Monma and Chris N. Potts. On the complexity of scheduling with batch setup times. *Oper. Res.*, 37(5):798–804, 1989.

[101] Clyde L. Monma and Chris N. Potts. Analysis of heuristics for preemptive parallel machine scheduling with batch setup times. *Oper. Res.*, 41(5):981–993, 1993.

[102] Thi Huyen Chau Nguyen, Werner Grass, and Klaus Jansen. Exact polynomial time algorithm for the response time analysis of harmonic tasks. In Cristina Bazgan and Henning Fernau, editors, *Proc. IWOCA 2022*, volume 13270 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2022.

[103] Martin Niemeier and Andreas Wiese. Scheduling with an orthogonal resource constraint. *Algorithmica*, 71(4):837–858, 2015.

[104] Timm Oertel, Joseph Paat, and Robert Weismantel. A colorful steinitz lemma with applications to block integer programs, 2022.

[105] Daniel R. Page and Roberto Solis-Oba. Makespan minimization on unrelated parallel machines with a few bags. *Theor. Comput. Sci.*, 821:34–44, 2020.

[106] Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

[107] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, 1982.

[108] Yves Pochet and Laurence A. Wolsey. *Production Planning by Mixed Integer Programming.* Springer Series in Operations Research and Financial Engineering, Berlin, Heidelberg, 2006.

[109] Frances Rosamond. Parameterized complexity-news. *The Newsletter of the Parameterized Complexity Community Volume*, 2006.

[110] Petra Schuurman and Gerhard J. Woeginger. Preemptive scheduling with job-dependent setup times. In Robert Endre Tarjan and Tandy J. Warnow, editors, *Proc. SODA 1999*, pages 759–767. ACM/SIAM, 1999.

[111] Omri Serlin. Scheduling of time critical processes. In *Proc. AFIPS 1972*, volume 40 of *AFIPS Conference Proceedings*, pages 925–932. AFIPS, 1972.

[112] Ernst Steinitz. Bedingt konvergente reihen und konvexe systeme. *Journal für die reine und angewandte Mathematik*, 143:128–176, 1913.

[113] Vitaly A. Strusevich. Approximation algorithms for makespan minimization on identical parallel machines under resource constraints. *J. Oper. Res. Soc.*, 72(9):2135–2146, 2021.

[114] Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.*, 37(4):1283–1311, 2019.

[115] Ken Tindell, Alan Burns, and Andy J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real Time Syst.*, 6(2):133–151, 1994.

[116] Wenxun Xing and Jiawei Zhang. Parallel machine scheduling with splitting jobs. *Discret. Appl. Math.*, 103(1-3):259–269, 2000.

[117] Minyi Yue. On the exact upper bound for the multifit processor scheduling algorithm. *Annals of Operations Research*, 24(1):233–259, 1990.

[118] Ming Zhao and Ismael R. de Farias Jr. The mixing-mir set with divisible capacities. *Math. Program.*, 115(1):73–103, 2008.

ERKLÄRUNG

Hiermit gebe ich folgende Erklärungen ab:

- Diese Abhandlung ist, abgesehen von der Beratung durch meinen Betreuer Klaus Jansen, nach Inhalt und Form meine eigene Arbeit. Ich habe sie eigenständig und nur mit den angegebenen Hilfsmitteln verfasst.
- Die Arbeit ist unter Einhaltung der Regeln guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft entstanden.
- Es wurde mir noch nie ein akademischer Grad entzogen.

Teile dieser Arbeit sind bereits an anderer Stelle im Rahmen eines Prüfungsverfahrens vorgelegt worden. Dies betrifft Kapitel 3, welches meine Masterarbeit [39] um neue Ergebnisse erweitert. Kein anderer Teil dieser Arbeit ist bereits an anderer Stelle im Rahmen eines Prüfungsverfahrens vorgelegt worden. Teile wurden, wie in der Arbeit gekennzeichnet, im Rahmen wissenschaftlicher Veröffentlichungen publiziert.

Kiel, November 2022

Max A. Deppert

ERKLÄRUNG ÜBER DEN EIGENANTEIL AN PUBLIKATIONEN MIT MEHREREN AUTOREN

Die Ideen und Beweise der drei Arbeiten [2, 3, 4] hat Herr Deppert zu mehr als der Hälfte ausgearbeitet und die Arbeiten fast vollständig selbstständig und alleine geschrieben. Bei der Erstellung der Manuskripte wurde er durch seinen Betreuer Klaus Jansen entsprechend unterstützt.

Die initiale Idee für die theoretische und praktische Arbeit [1] stammt von Lars Rohwedder. Herr Deppert hat weitere Optimierungsideen entwickelt und war insbesondere für die Implementierung und Analyse verantwortlich. Der Aufschrieb wurde gleichmäßig aufgeteilt.

Bei der Arbeit [5] hat Herr Deppert zu gleichen Teilen die Ideen und Beweise entwickelt und ausgearbeitet. Ebenso wurde der Aufschrieb gleichmäßig aufgeteilt.

Kiel, November 2022

Max Amadeus Deppert

Kiel, November 2022

Klaus Jansen

COLOPHON

This document was typeset using `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Thank you very much for your feedback and contribution.

Final Version as of April 24, 2023 (classicthesis v4.6).