

DAVID SIMON FISCHER

APPROXIMATION AND PARAMETERIZED
ALGORITHMS FOR SCHEDULING PROBLEMS

APPROXIMATION AND PARAMETERIZED ALGORITHMS FOR
SCHEDULING PROBLEMS

DAVID SIMON FISCHER

DISSERTATION

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
der Technischen Fakultät der Christian-Albrechts-Universität zu Kiel
Eingereicht im September 2025

1. GUTACHTEN
2. GUTACHTEN

Prof. Dr. Klaus Jansen, Christian-Albrechts-Universität zu Kiel
Prof Dr. José Verschae, Pontificia Universidad Católica de Chile

DATUM DER DISPUTATION 05.12.2025

“Either mathematics is too big for the human mind, or the human mind is more than a machine.”

— Kurt Gödel

Per aspera ad astra

ABSTRACT

In this thesis, we give new algorithmic results for $\mathcal{NP}$ -hard scheduling problems in the fields of approximation and parameterization. In the first part, we give constant-factor approximation algorithms for two scheduling problems. The first is the problem of *Coupled Task Scheduling*, where we are given n jobs to be scheduled on a single machine, and each job j consists of two separate tasks a_j and b_j , and a delay time L_j . In any schedule, the second task b_j has to be scheduled to start exactly L_j time units after the first task a_j finishes. We aim to minimize the sum of completion times. We give constant-factor approximation algorithms for various $\mathcal{NP}$ -hard variants of this general problem, and prove $\mathcal{NP}$ -hardness of some further variants. The second of these problems is usually referred to as *Preemptive Scheduling with Batch Setup Times*, where we are given n jobs that are partitioned into c classes, to be scheduled on m identical parallel machines while allowing preemption, where a setup time s_i has to be scheduled when switching classes on a machine. The goal is to minimize the makespan. We give a $(4/3 + \varepsilon)$ -approximate algorithm with run time in $\mathcal{O}(n^2 \log(1/\varepsilon))$ for any $\varepsilon > 0$, which improves upon the previously best known approximation ratio of $3/2$.

In the second part, we exclusively look at the problem of *Scheduling on Uniform Machines*, but with different objective functions ($C_{\max}$, $C_{\min}$ and C_{envy}), in different input settings (offline and online) and from different angles (parameterization and approximation). In the parameterized setting, we consider parameters $p_{\max}$ (largest processing time occurring) and d (number of different job types). We first focus on parameterization and the objective C_{envy} . For both the jobs n and the machines m encoded in high-multiplicity, we give an algorithm for this objective parameterized only in $p_{\max}$ and d . Such an algorithm was not known before, and it already matches the best-known run times for objectives $C_{\max}$ and $C_{\min}$ for this encoding from the literature. We also subsequently manage to give an algorithm with improved run time, matching the only recently discovered, best-possible run times for these parameters under the ETH for both $C_{\max}$ and $C_{\min}$. We continue our research by turning to an *online* version of this problem, where jobs are inserted into or removed from an instance over time, and objectives $C_{\max}$ and $C_{\min}$. Here, for both these objective functions, we combine parameterization and approximation in order to obtain the first algorithm in this setting and machine environment with approximation error of only ε , for any $\varepsilon > 0$. We manage to get rid of the parameterization altogether when allowing an additional load error of $\mathcal{O}(\varepsilon p_{\max})$.

ZUSAMMENFASSUNG

In dieser Arbeit werden neue algorithmische Ergebnisse für $\mathcal{NP}$ -schwere Scheduling-Probleme in den Bereichen Approximation und Parametrisierung gezeigt. Im ersten Teil werden Algorithmen mit konstantem Approximationsfaktor für zwei Scheduling-Probleme vorgestellt. Das Erste dieser Probleme ist das sogenannte *Coupled Task Scheduling*, bei dem n Jobs auf einer einzelnen Maschine einzuplanen sind, wobei jeder Job j aus zwei separaten Tasks a_j und b_j sowie einer Verzögerungszeit L_j besteht. In jedem gültigen Ablaufplan muss der Start des zweiten Tasks b_j genau L_j Zeiteinheiten nach dem Ende des ersten Tasks a_j erfolgen. Das Ziel dabei ist, die Summe der Fertigstellungszeitpunkte zu minimieren. Wir geben Algorithmen mit konstantem Approximationsfaktor für verschiedene $\mathcal{NP}$ -schwere Varianten dieses Problems an, und beweisen die $\mathcal{NP}$ -Schwere einiger weiterer Varianten. Das Zweite der hier betrachteten Probleme wird üblicherweise als *Preemptive Scheduling with Batch Setup Times* bezeichnet. Dabei müssen n Jobs, die in c Klassen unterteilt sind, auf m identischen parallelen Maschinen eingeplant werden, wobei Unterbrechungen der Jobs erlaubt sind und beim Wechsel der Klasse auf einer Maschine eine Rüstzeit s_i eingeplant werden muss. Hier ist das Ziel, den Makespan zu minimieren. Wir geben einen $(4/3 + \varepsilon)$ -Approximationsalgorithmus mit einer Laufzeit von $\mathcal{O}(n^2 \log(1/\varepsilon))$ für jedes $\varepsilon > 0$ an, der damit eine allgemein bessere Lösungsgüte als der bisher beste bekannte Algorithmus mit einem Approximationsgüte von $3/2$ liefert.

Im zweiten Teil wird ausschließlich das Problem des *Scheduling on Uniform Machines* betrachtet, jedoch mit verschiedenen Zielfunktionen ($C_{\max}$, $C_{\min}$ und C_{envy}), in unterschiedlichen Eingabeumgebungen (offline und online), und aus verschiedenen Blickwinkeln (Parametrisierung und Approximation). Bei der Parametrisierung werden die Parameter $p_{\max}$ (größte auftretende Ausführungszeit eines Jobs) und d (Anzahl unterschiedlicher Jobtypen) betrachtet. Zunächst fokussieren wir uns auf Parametrisierung und die Zielfunktion C_{envy} . Für den Fall, dass sowohl die Jobs n als auch die Maschinen m in High-Multiplicity kodiert sind, geben wir einen Algorithmus für diese Zielfunktion an, der nur bezüglich $p_{\max}$ und d parametrisiert ist. Ein solcher Algorithmus war bisher nicht bekannt, und unser Algorithmus besitzt gleichzeitig bereits dieselbe Laufzeit wie die schnellsten, in der Literatur bekannten Algorithmen für die verwandten Zielfunktionen $C_{\max}$ und $C_{\min}$ unter Verwendung derselben Kodierung. Anschließend geben wir einen Algorithmus mit verbesserter Laufzeit an, der dieselbe Laufzeit wie die erst kürzlich gefundenen Algorithmen, mit unter der ETH bestmöglichen Laufzeiten für diese Parameter für $C_{\max}$ und $C_{\min}$,

erreicht. Wir wenden uns danach einer *Online*-Version dieses Problems zu, bei der Jobs im Laufe der Zeit zu einer Instanz hinzugefügt oder aus ihr entfernt werden, und betrachten dabei die Zielfunktionen $C_{\max}$ und $C_{\min}$. Hier kombinieren wir für beide Zielfunktionen die Ideen der Parametrisierung und Approximation, um den ersten Algorithmus mit einem Approximationsfehler von nur ε für beliebiges $\varepsilon > 0$ in dieser Eingabe- und Maschinenumgebung zu erhalten. Wir schaffen es, die Parametrisierung vollständig zu eliminieren, sofern ein zusätzlicher Lastfehler von $\mathcal{O}(\varepsilon p_{\max})$ erlaubt wird.

PUBLICATIONS

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

- [1] Hauke Brinkop, David Fischer, and Klaus Jansen. Robust scheduling on uniform machines - new results using a relaxed approximation guarantee. In *Proc. WAOA 2025*, pages 111–126. Springer, 2026.
- [2] David Fischer and Péter Györgyi. Approximation algorithms for coupled task scheduling minimizing the sum of completion times. *Ann. Oper. Res.*, 328(2):1387–1408, 2023.

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

- [3] Hauke Brinkop, David Fischer, and Klaus Jansen. High multiplicity scheduling on uniform machines in fpt-time. *CoRR*, abs/2203.01741, 2022.
- [4] Max A. Deppert, David Fischer, and Klaus Jansen. A $(4/3 + \varepsilon)$ -approximation for preemptive scheduling with batch setup times. *CoRR*, abs/2508.14528, 2025. In submission.

Additionally, [Section 4.4](#) is based on yet unpublished, joint work with Hauke Brinkop, Klaus Jansen, Björn Schumacher, Felix Ohnesorge, and Lis Piroton.

*In nova fert animus mutatas dicere formas corpora.
Di, coeptis - nam vos mutastis et illas -
adspirate meis primaque ab origine mundi
ad mea perpetuum deducite tempora Carmen!*

— Ovid, *Metamorphoses*

ACKNOWLEDGMENTS

I can only describe the process of doing a PhD as a transformative experience, both professionally and personally. Responsible for this transformation are not solely the things I learned and experiences I had, but the people I met along the way.

First of all, I want to thank my advisor Klaus Jansen for his support, his encouragement and his guidance, without which writing this thesis would not have been possible for me. I also sincerely want to thank Thomas Wilke for hosting me during my time in Kiel. Furthermore, I owe much gratitude to Florian Jaehn for hosting me at the HSU during the last year of my studies and supporting all my endeavors. Special thanks also goes out to Max Amadeus Deppert, who always believed in me and my ideas, and really helped me find my footing in Kiel. To my co-authors Hauke Brinkop and Péter Györgyi, I am very grateful for our fun, joint work and fruitful discussions.

I had many stops during my time as a PhD student. At each stop, I was lucky enough to spend my time with some remarkable people. At TUHH, I had the pleasure to meet Laura Codazzi, Esther Galby, Julian Golak, Matthias Kaul, Christian Ortlieb, Jens M. Schmidt, Tobias Stamm, and Antoine Wiehe. At Kiel University, I had the pleasure to meet Annika Huch, Ute Iaquinto, Kai Kahler, Parvaneh Karimi-Massouleh, Lis Piroton, Janina Reuter, Björn Schumacher, Malte Tutas, Corinna Wambsganz, and Felix Ohnesorge. Finally, at HSU, I had the pleasure to meet Nicolas Fauve, Julius Hardt, Andreas Hipp, Katrin Hofmann, Dominik Kreß, Finn Meissner, Simone Neumann, Lara Nehrke, Hendrik Schanz, Wachiraya Udomyat, and Frank Wiedra. I am sincerely grateful to have met, worked with and got to know each and every one of you.

Last but not least, I want to thank my girlfriend, my family and my friends, in Hamburg, Würzburg, Kiel and anywhere else in the world, who were always there for me during all the different stages of my PhD student journey.

CONTENTS

1	Introduction	1
1.1	General Preliminaries	3
1.2	Overview	11
1.2.1	Constant Factor Approximation Algorithms . .	11
1.2.2	Parameterized Algorithms and Approximation Schemes	12
I	Constant Factor Approximation Algorithms	
2	Coupled Task Scheduling Minimizing the Sum of Completion Times	17
2.1	Introduction	17
2.1.1	Related Work	19
2.1.2	Our Contribution	21
2.2	Complexity Results	23
2.3	Approximation Results	28
2.3.1	Lower Bounds on the Optimum	28
2.3.2	CTP with Fixed Processing Times	29
2.3.3	CTP with Fixed Delay Times	32
2.3.4	CTP with Related Processing and Delay Times .	36
2.4	Bi-Objective Approximation	37
2.5	Conclusion	38
3	Preemptive Scheduling with Batch Setup Times	41
3.1	Introduction	41
3.1.1	Related Work	41
3.1.2	Our Contribution	42
3.2	Preliminaries	43
3.2.1	Structuring an Instance.	43
3.2.2	Batch Wrapping	44
3.3	Technical Overview	46
3.4	A $\frac{4}{3}$ -Approximation for Nice Instances	47
3.5	Structural Insights on General Instances	52
3.5.1	Non-Nice Classes with Multiple Setup Times .	53
3.5.2	Constructing a Nice Sub-Instance	53
3.5.3	Combining Classes of Type Eight and Nine . . .	56
3.5.4	Classes of Type Eight Remain	57
3.5.5	Classes of Type Nine Remain	67
3.6	The Approximation Algorithm	72
3.6.1	Classes of Type Eight Remain	73
3.6.2	Classes of Type Nine Remain	77
3.7	Conclusion	80

II Parameterized Algorithms and Approximation Schemes

4	FPT Algorithms for Envy Minimization on Uniform Machines	85
4.1	Introduction	85
4.1.1	Related Work	86
4.1.2	Our Contribution	87
4.2	Preliminaries	87
4.3	$Q HM(m,n) C_{\text{envy}}$ Parameterized by $p_{\max}$	88
4.4	Achieving ETH-Tight Run Time for $Q HM(m,n) C_{\text{envy}}$	92
4.5	Conclusion	102
5	Robust Scheduling on Uniform Machines	103
5.1	Introduction	103
5.1.1	Related Work	105
5.1.2	Our Contribution	106
5.2	Preliminaries	109
5.2.1	Sensitivity of ILPs	109
5.2.2	Lexicographically Minimal Vectors	110
5.2.3	An Exact Approach	111
5.3	Technical Overview	113
5.3.1	Construction of the Parameterized Robust Algorithm	114
5.3.2	Extension to Robust $Q C_{\min}$	122
5.3.3	Getting Rid of the Parameterization	123
5.4	Technical Details for Robust $Q C_{\max}$	124
5.4.1	Some Preliminaries	124
5.4.2	Permanent Case: Handling Insertion	127
5.4.3	Dynamic Case: Handling Removal	133
5.5	Technical Details for Robust $Q C_{\min}$	143
5.5.1	Some Preliminaries	143
5.5.2	Handling Removal	146
5.5.3	Handling Insertion	151
5.6	Preprocessing and Rounding an Instance	160
5.7	A (Dynamic) Grouping Technique	165
5.8	Robust Scheduling With Additive Error $p_{\max}$	172
5.9	Solving the ILPs	175
5.10	The Approximation Schemes	178
5.11	Construction of Legacy Schedules	180
5.12	Conclusion	181
6	Bibliography	183

ACRONYMS & ABBREVIATIONS

CTP	Coupled Task Processing
EPAS	Efficient Parameterized Approximation Scheme
EPTAS	Efficient Polynomial-Time Approximation Scheme
ETH	Exponential Time Hypothesis
ILP	Integer Linear Program
FPT	fixed-parameter tractable
FPTAS	Fully Polynomial-Time Approximation Scheme
LHS	left-hand side
LP	Linear Program
PTAS	Polynomial-Time Approximation Scheme
RHS	right-hand side
XP	slice-wise polynomial
cf.	compare (confer)
e.g.	for example (exempli gratia)
et al.	and others (et alia)
i.e.	that is (id est)
lex. min.	lexicographically minimal
s.t.	subject to
wlog.	without loss of generality
w.r.t.	with respect to

INTRODUCTION

Even if we don't necessarily recognize them as such, scheduling problems in one form or another occur every single day for most, if not all, of us. If, for example, you are given a list of household chores, and are wondering in which order best to complete them, you are engaging in scheduling. Computing the "solution", i. e. in which order to complete the chores, to this "problem" seems rather easy. If you are living now in a flat-share with another person, and you want to divide up the chores in a way that is fair in some sense, the problem becomes harder, but may still be somewhat straightforward and intuitively to solve optimally. What if you are living in a three-person flat-share? What about four? This example is also by far not the only natural occurrence: Scheduling problems can be found in the management sciences, in industrial planning, in the process planning of any computer chip, and many more areas and fields.

In the most general sense, a *scheduling problem* refers to the problem of assigning a number n of *jobs* to a number m of *machines* with the goal of minimizing or maximizing some *objective function*. Each job has a *processing time*, and both the jobs and machines may have some additional *characteristics*. The assignment of jobs to machines is called a *schedule*. The notion of jobs and machines, as stated here, is mostly an abstraction of some real world activities and available resources. In the example above, jobs may refer to the list of household chores, and the people living in the flat can be modeled as a machine each. Personal preferences for specific chores can then be seen as characteristics of a machine, and the objective function is to maximize the fairness of task allocation between all flat mates. This kind of abstraction is very useful for devising and reusing solution approaches: If you find a way to solve the "household chore problem" in its form of jobs and machines, you may use this solution approach for all other problems you encounter that can be modeled in the same way using jobs and machines. For a comprehensive introduction into the topic of scheduling theory, we refer the reader to "Elements of Scheduling" by Lenstra and Shmoys [96].

Even before the advent of computers, people have used *algorithms*, finite sequences of clearly defined instructions that return some output when applied to some input, to solve real-world mathematical problems, or abstractions thereof. An early example of an algorithm that is still in use today is Euclid's algorithm for finding the greatest common divisor of two integers, first described around 300 BC [114]. Any finite sequence of instructions, as defined above, that you de-

wise for returning a schedule to the household chore problem is, in the general sense, an algorithm for this problem. The input usually is an instance of a problem, an assignment of actual values to the problem parameters, and the output a solution for this instance. For our household chore problem, the input would be a specific value for the number of jobs n and machines m , and the output a schedule for these values. The run time of this algorithm then is the time it takes you to perform all the instructions on the input to obtain the output. With the emergence of computers, as formally defined by the concept of the Turing machine, the concept of algorithms and their run time became more formalized as well. Any algorithm can be modeled as a Turing machine, and the run time of an algorithm for an instance I of a problem is formally measured in the number of operations it takes its corresponding Turing machine to compute a solution as a function in the encoding size $|I|$ of I . Mostly, but not always, the encoding of an instance is done in bits; $|I|$ then equals the number of bits necessary to store I .

An algorithm is called *efficient* if its run time can be expressed as a polynomial function in $|I|$, for all possible instances I of a problem, as such algorithms can be expected to run in reasonable time on modern computers. The natural question then arises: Are there efficient algorithms for all mathematical problems? And furthermore, maybe not as intuitively natural: Can we divide problems into classes of different hardness w. r. t. to the efficiency of their algorithms? These questions are, among others, the subject of the field of complexity theory. For our purposes, the most important complexity classes are the classes $\mathcal{P}$ and $\mathcal{NP}$. In $\mathcal{P}$, there are contained all problems that do allow efficient, polynomial time algorithms for finding a solution. In $\mathcal{NP}$, there are contained all problems that allow efficient, polynomial time algorithms for verifying a given solution. This directly implies $\mathcal{P} \subseteq \mathcal{NP}$. It is however unclear if this relation is reversible, i. e. if $\mathcal{NP} \subseteq \mathcal{P}$ and thus $\mathcal{P} = \mathcal{NP}$, or not, and thus $\mathcal{P} \neq \mathcal{NP}$. Even though the latter is conjectured to be true [35], proving either statement conclusively would have wide reaching consequences in the fields of algorithms, complexity and cryptography. As a way to identify problems that are at least as hard as the hardest problems in $\mathcal{NP}$, and thus not solvable in polynomial time should $\mathcal{P} \neq \mathcal{NP}$ turn out to be true, the notion of $\mathcal{NP}$ -hardness was introduced. Problems that are both $\mathcal{NP}$ -hard and in $\mathcal{NP}$ are called $\mathcal{NP}$ -complete. For a general introduction into this area of complexity theory, we refer the reader to “Computers and Intractability: A Guide to the Theory of NP-Completeness” by Garey and Johnson [55].

For now, we do not know if we can solve $\mathcal{NP}$ -complete problems in polynomial time. It is even conjectured that the run time of any algorithm for such problems must be at least roughly exponential in the encoding size of an instance, as stated in the famous Exponential

Time Hypothesis (ETH) by Impagliazzo and Paturi [73]. As it turns out, many real-world problems, including a multitude of scheduling problems, are $\mathcal{NP}$ -complete. Does this however mean we should just give up on trying to solve these problems? The answer to this question from the relevant literature is a resounding “No!”. Even though we may only find exponential time algorithms to solve these problems exactly, i. e. to obtain an optimal solution, for small instances, computers may still find such solutions in reasonable time. And what if we find some properties that make these problems that hard to solve, and then devise algorithms that are still able to find optimal solutions, but have run time exponential only in these parameters? Instances where such parameters are comparatively small then can still be solved efficiently. And what if we relax our goal of solving problems optimally, and instead try to obtain solutions that are not optimal, but close to optimal and computable in reasonable time? In this thesis, we focus on solving $\mathcal{NP}$ -complete scheduling problems algorithmically, a task that we mainly tackle with the latter two of the three mentioned approaches, formally known as *parameterization* and *approximation*.

1.1 GENERAL PRELIMINARIES

For readability purposes, here we introduce some preliminaries and notations used in multiple chapters of this work.

NOTATION Throughout this work, we will use the classic three-field $\alpha|\beta|\gamma$ notation system for scheduling problems as introduced by Graham et al. [57], with α representing the machine environment, β representing the characteristics of the jobs, and γ representing the objective function. We use the asymptotic notation $\mathcal{O}(\cdot)$, $\mathcal{o}(\cdot)$, $\Omega(\cdot)$, $\omega(\cdot)$ and $\Theta(\cdot)$ to define asymptotic bounds of functions in the common way found in the literature, cf. Cormen et al. [37, 3.1 Asymptotic notation]. Guessing a value x from a discrete, finite set of values X for some procedure means to execute the procedure for all values in $|X|$ until some $x \in X$ with the desired properties is found. We write $\text{poly}(x_1, \dots, x_n)$ to refer to any polynomial with variables $x_1, \dots, x_n$ for some $n \in \mathbb{N}_{\geq 0}$. By $\log$, we refer to the binary logarithm $\log_2$. By $|I|$, we refer to the (binary) encoding size of an instance I of a specified problem, and by $\langle c_1, \dots, c_n \rangle$, we refer to the (binary) encoding size $\log(c_1) + \dots + \log(c_n)$ of numbers $c_1, \dots, c_n$ for some $n \in \mathbb{N}_{\geq 0}$. We assume wlog. that for any occurring $\varepsilon > 0$, ε^{-1} is integral. Let $\mathbf{e}^{(j)}$ denote the j -th unit vector, i.e. $\mathbf{e}_j^{(j)} = 1$ and $\mathbf{e}_{j'}^{(j)} = 0$ for all $j' \neq j$. We use the Bourbaki interval notation, e. g. write $[0; 1)$ for $\{x \mid 0 \leq x < 1\}$. We extend this notation to integer intervals, e. g. write $(0..4]$ for $\{1, 2, 3, 4\}$. As shorthand, we use $[x] := [1..x]$.

DECISION VS OPTIMIZATION PROBLEMS Computational problems, that is problems that ask for a solution via an algorithm, can take different forms. The two most important forms of problems for this thesis are *decision problems* and *optimization problems*. A decision problem Π is a problem whose solution is always either “yes” or “no”, and an instance of I of Π is *accepted* if an algorithm that solves the problem outputs “yes”, and *rejected* if the algorithm outputs “no”. An example of a decision problem is the problem of deciding for some given number x if x is a prime number. In contrast, an optimization problem Π is a problem that has possibly many *feasible* solutions which can be the output of an algorithm. A solution is feasible as long as it conforms to the definition of feasibility as given by the problem statement. The goal then becomes not to find some feasible solution, but a best solution among all feasible solutions, which is then called an *optimal* solution. A solution is optimal if it either minimizes or maximizes some objective function f over all feasible solutions, depending on the stated problem. The former type of optimization problems are referred to as *minimization problems*, the latter as *maximization problems*.

While $\mathcal{NP}$ -completeness is only formally defined for decision problems, any optimization problem Π can be transformed into a corresponding decision problem Π' by introducing some bound T on the value of solutions, and formulating Π' as the question: “Is there a solution to Π with objective value at most T (at least T)?”, if Π is a minimization (maximization) problem. This immediately implies that Π is computationally at least as hard as Π' . In this thesis, we mostly deal with optimization problems. If we refer to an optimization problem as $\mathcal{NP}$ -hard or prove its $\mathcal{NP}$ -hardness, by that we refer technically to the $\mathcal{NP}$ -hardness of its corresponding decision problem. Many optimization problems can also be solved by repeatedly solving the corresponding decision problem for different values of T , until the best value for T is found for which an instance is still accepted. The run time of such an algorithm then is the run time of the algorithm for solving the decision problem times the number of iterations it takes to find the optimal value for T .

INTEGER PROGRAMMING Integer (Linear) Programming is a powerful tool in the world of solving optimization problems, as many such problems can be expressed as Integer Linear Programs (ILPs). The basic form of an ILP for minimization problems is:

$$\begin{aligned} &\min c^\top x \\ &\text{s.t.} \\ &Ax \leq b \\ &\text{var } x \in \mathbb{N}_{\geq 0} \end{aligned}$$

Here, $x \in \mathbb{N}_{\geq 0}$ are variables, $A \in \mathbb{Z}^{M \times N}$ is a constraint matrix, $c \in \mathbb{Z}^N$ an objective vector, and $b \in \mathbb{Z}^M$ the right-hand side (RHS) of the ILP,

for some integers $M, N \in \mathbb{N}_{\geq 0}$. The goal is to find a solution for x , subject to $Ax \leq b$, such that $c^\top x$ is minimized. For maximization problems, “min” is replaced by “max”, and the goal is to find a solution such that $c^\top x$ is maximized. We refer the reader to “Integer Programming” by Conforti et al. [34] for a general introduction into this topic. The problem of solving such ILPs is generally $\mathcal{NP}$ -hard, but a series of algorithms which depend exponentially only on either M or N have been developed, as solving the ILP formulation of some optimization problem is akin to solving the problem itself. Such algorithms are particularly useful when the values of either M or N are relatively small. The currently fastest known algorithm exponentially dependent on M is by Jansen and Rohwedder [81], with a run time of $\mathcal{O}(\sqrt{M}\|A\|_\infty)^{2M} \log^2(\|b\|_\infty) + \mathcal{O}(MN)$, where $\sqrt{M}\|A\|_\infty$ stems from an upper bound on the hereditary discrepancy of the constraint matrix A . Considering N , the currently fastest known algorithm is by Kannan [83], with a run time of $2^{\mathcal{O}(N \log N)} |I|^{\mathcal{O}(1)}$, where $|I|$ is the encoding length of the ILP.

Notably, if the constraint of the variables to integers is dropped, i. e. we allow $x \in \mathbb{R}_{\geq 0}$ instead of $x \in \mathbb{N}_{\geq 0}$, the respective formulation can be solved in polynomial time in the size of the input [34]. Such a form is called Linear Program (LP), and is often referred to as the *LP-relaxation* of the corresponding ILP. While an LP in general does not model the same problem as is modeled by the corresponding ILP, solving the LP-relaxation is useful to get a lower bound (in the case of minimization) or an upper bound (in the case of maximization) on the objective value of the ILP: Every feasible solution of an ILP is also a feasible solution of the LP-relaxation, implying that the objective value of the LP can only be better than that of the corresponding ILP.

N -FOLD INTEGER PROGRAMMING In this work, we make use of known results for N -fold ILPs, which are a specific type of block structured ILPs. More precisely, N -fold ILPs are ILPs with a special, block structured constraint matrix $\mathcal{A}$ of total size $(R + SH) \times NH$ for $R, S, H, N \in \mathbb{N}_{\geq 0}$ consisting of R arbitrary rows, with a block diagonal matrix below where each block has size $S \times H$. $\mathcal{A}$ can also be described as consisting of block matrices $A^{(i)} \in \mathbb{Z}^{R \times H}$, also called the linking constraints, and block matrices $B^{(i)} \in \mathbb{Z}^{S \times H}$, also called the local constraints, for each $i \in [N]$, with $\mathcal{A}$ being composed as follows:

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

Such ILPs can be solved with an algorithm by Cslovjecsek et al. [38].

Lemma 1 (Cslovjecsek et al. [38]). *An optimal solution to any N -fold ILP can be computed in time $2^{\mathcal{O}(RS^2)}(RS\|\mathcal{A}\|_\infty)^{\mathcal{O}(R^2S+S^2)}(NH)^{1+o(1)}$.*

SCHEDULING ON UNIFORM MACHINES In this work, we consider, among others, scheduling problems on uniform machines (sometimes also referred to as uniformly related machines), where we are given a set $\mathcal{J}$ of n jobs and m uniform machines, where each job J has a processing time $p_J \in \mathbb{N}_{\geq 0}$, and each machine i has a speed $s_i \in \mathbb{Q}_{>0}$, and a job J needs time $\frac{p_J}{s_i}$ to be completed on a machine i . This machine setting is denoted as Q in the α field of the three-field notation. A schedule $\sigma: \mathcal{J} \rightarrow [m]$ is an assignment of jobs to machines. Let $C_i^{(\sigma)}$ denote the time when machine i is finished processing the jobs assigned to it by a schedule σ , i. e. its completion time under σ . The load L_i^σ of a machine i in a schedule σ is the sum of all processing times assigned to i by σ , i. e. $L_i^\sigma = \sum_{J \in \mathcal{J}: \sigma(J)=i} p_J$. If σ is clear from context, we may omit it. For scheduling problems in general, we use the terms “schedule” and “solution” interchangeably.

Regarding uniform machines, we will look at three objective functions: $C_{\max}$, that is, find σ such that $\max_i C_i^{(\sigma)}$ is minimal; $C_{\min}$, that is, find σ such that $\min_i C_i^{(\sigma)}$ is maximal; and C_{envy} , that is, find σ such that $\max_i C_i^{(\sigma)} - \min_i C_i^{(\sigma)}$ is minimal. For $Q||C_{\max}$, the goal is then to find a schedule σ such that the makespan, the maximum completion time of any machine in σ , is minimized. For $Q||C_{\min}$, the goal is to find a schedule σ such that the minimum completion time of any machine in σ is maximized. For $Q||C_{\text{envy}}$, the goal is to find a schedule σ such that the difference between the makespan and the minimum completion time of any machine in σ is minimized.

Sometimes, we will use a different encoding of scheduling instances on uniform machines, commonly referred to as *high-multiplicity* encoding. Let $p \in \mathbb{N}_{\geq 0}^d$ be the vector of (different) job processing times and $v \in \mathbb{N}_{\geq 0}^d$ the corresponding multiplicity vector, i. e. for $j \in [d]$ there are v_j jobs of processing time p_j in the instance. For this encoding, we redefine a schedule σ to be a map $\sigma: [m] \times [d] \rightarrow \mathbb{N}_{\geq 0}$, mapping a machine $i \in [m]$ and a job type $j \in [d]$ to the number of jobs of type j that are scheduled on a machine i , such that $\sum_{i \in [m]} \sigma(i, j) = v_j$ for all $j \in [d]$. By $\sigma(i)$, we refer to the d -dimensional vector of all jobs scheduled on a machine i in σ , i. e. $\sigma_j(i) = \sigma(i, j)$. We refer to this encoding as $HM(n)$. Analogously, let $s \in \mathbb{Q}^\tau$ be the vector of machine speeds and $\mu \in \mathbb{N}_{\geq 0}^\tau$ the corresponding multiplicity vector. If both jobs and machines are encoded in high-multiplicity, we refer to the encoding as $HM(m, n)$.

Note that for either high-multiplicity encoding definition, jobs of the same type have become indistinguishable, and for $HM(n)$, we now have $\text{poly}(|I|) = \text{poly}(\log n, d, m)$, and for $HM(m, n)$, $\text{poly}(|I|) = \text{poly}(\log n, \log m, \tau, d)$, for any such encoded instance I . This is in contrast to the individual encoding of the n jobs by their processing

time, and of the m machines by their speed, also referred to as *low-multiplicity*. Also note that any low-multiplicity encoded instance I can be transformed into a $HM(n)$ encoded one, and any $HM(n)$ encoded instance I can be transformed into a $HM(m, n)$ encoded one, in $\mathcal{O}(\text{poly}(|I|))$ time each. When using high-multiplicity encoding, we always specify which by either stating so before, or by including either $HM(n)$ or $HM(m, n)$ in the β -field of the respective three-field notation, i. e. $Q|HM(m, n)|C_{\max}$ refers to the problem of makespan minimization on uniform machines where both jobs and machines are encoded in high-multiplicity.

A helpful tool in formulating ILPs for the considered scheduling problems are *configurations*, i. e. all ways to place jobs with total processing time at least some lower bound ℓ and at most some upper bound u , which we define formally here.

Definition 1 (Configurations).

$$\mathcal{C}: \mathbb{Q}_{\geq 0} \times \mathbb{Q}_{\geq 0} \rightarrow \mathbb{P}(\mathbb{N}_{\geq 0}^d), \quad \ell, u \mapsto \left\{ c \in \mathbb{N}_{\geq 0}^d \mid \ell \leq p^\top c \leq u \right\}.$$

If only one parameter is given for $\mathcal{C}$, we assume it to be u , and simultaneously assume $\ell = 0$. The two most common ILP formulations to decide feasibility of a makespan T for $Q||C_{\max}$, both of which can easily be adapted to decide feasibility of a minimum completion time for $Q||C_{\min}$, are the [Configuration-ILP](#) and [Assignment-ILP](#) and defined as follows. Here, the former is defined using $HM(m, n)$ encoding, the latter using $HM(n)$ encoding.

Configuration-ILP

$$\begin{aligned} \sum_{t \in [\tau]} \sum_{c \in \mathcal{C}(s_t T)} x_{t,c} \cdot c &= v \\ \sum_{c \in \mathcal{C}(s_t T)} x_{t,c} &= \mu_t & \forall t \in [\tau] \\ \text{var } x_{t,c} &\in \mathbb{N}_{\geq 0} & \forall t \in [\tau], c \in \mathcal{C}(s_t T) \end{aligned}$$

Assignment-ILP

$$\begin{aligned} \sum_{i \in [m]} x_{i,j} &= v_j & \forall j \in [d] \\ \sum_{j \in [d]} p_j x_{i,j} &\leq \lfloor s_i T \rfloor & \forall i \in [m] \\ \text{var } x_{i,j} &\in \mathbb{N}_{\geq 0} & \forall (i, j) \in [m] \times [d] \end{aligned}$$

Both ILP formulations have N -fold structure; we discuss the specific block parameters in the appropriate places. The parameter d is commonly used to refer to the number of different job processing times

even in low-multiplicity encoding; the parameter $p_{\max}$ refers to the largest processing time occurring in an instance. Note that $d \leq p_{\max}$. Both these parameters are highly relevant not only in the literature, but also for us when looking at scheduling problems on uniform machines from the perspective of “parameterization”.

PARAMETERIZATION Unless $\mathcal{P} = \mathcal{NP}$, one cannot hope to solve $\mathcal{NP}$ -hard optimization problems optimally in polynomial time. However, while this is true for general instances of all $\mathcal{NP}$ -hard problems, there may be one or more “parameters” describing the part of the problem that make it hard to solve. The hope is to then obtain algorithms whose run time is super-polynomial only in these parameters, and polynomial in the encoding size of the rest of the input. For general instances, the parameters may themselves be polynomial in the input size, but instances where these parameters are small may then be solved efficiently (in polynomial time). This idea of measuring the run time of an algorithm for some problem not solely in terms of the total encoding size of an instance $|I|$, but also some problem specific parameter(s), lead to the notion of *fixed-parameter tractability* and the theory of parameterization in general, as first systematically described by Downey and Fellows [43].

Definition 2 (Parameterized Problems). Let Π be some problem, k some parameter, and $f : \mathbb{N}_{\geq 0} \mapsto \mathbb{N}_{\geq 0}$ some computable function.

- Π is called slice-wise polynomial (XP) w. r. t. parameter k if there exists an algorithm $\mathcal{A}$ such that $\mathcal{A}$ computes an optimal solution for Π in time $|I|^{f(k)}$ for any instance I of Π .
- Π is called fixed-parameter tractable (FPT) w. r. t. parameter k if there exists an algorithm $\mathcal{A}$ such that $\mathcal{A}$ computes an optimal solution for Π in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for any instance I of Π .

The algorithm $\mathcal{A}$ is called XP or FPT algorithm with parameter k for Π , respectively, or simply a parameterized algorithm.

Note that k can be a combination of multiple parameters, i. e. $k = k_1 + \dots + k_c$ for some $c \in \mathbb{N}_{\geq 0}$. Both XP and FPT also refer to the respective complexity classes that contain all problems that allow XP and FPT algorithms. This definition implies that $\text{FPT} \subseteq \text{XP}$. To give an example, $Q||C_{\max}$ is FPT w. r. t. parameter $p_{\max}$ [90]. For an introduction into fixed-parameter tractability and parameterized complexity, we refer the reader to “Parameterized Algorithms” by Cygan et al. [39].

APPROXIMATION Another way to deal with hard problems is the idea to find solutions that are not necessarily optimal, but “good enough”, especially if this means keeping the computational expense low, i. e. obtain such solutions in polynomial time. At the same time, there is a natural desire to mathematically define what we mean by

“good enough” and how this can be measured, e. g. by giving some guarantee on how much the objective value of our solution differs from the optimum. This is where approximation algorithms come into play.

Definition 3 (α -Approximation). Let Π be some optimization problem, and let $\text{OPT}_\Pi(I)$ denote the optimal objective value of an instance I of Π . An algorithm $\mathcal{A}$ is an α -approximation algorithm for Π , if, for every instance I of Π

- $\mathcal{A}(I) \leq \alpha \text{OPT}_\Pi(I)$,

and Π is a minimization problem, or

- $\mathcal{A}(I) \geq \alpha \text{OPT}_\Pi(I)$,

and Π is a maximization problem, where $\mathcal{A}(I)$ denotes the objective value computed by solving I with $\mathcal{A}$. We also call α the *approximation factor* of algorithm $\mathcal{A}$.

Note that, while this definition does not require the run time of $\mathcal{A}$ to be polynomial, most approximation algorithms for $\mathcal{NP}$ -complete problems in the literature do have polynomial, or, as in some cases, super-polynomial but sub-exponential run time. This is simply due to the fact that, when allowing exponential run time, all $\mathcal{NP}$ -complete problems can be solved optimally (i. e. with approximation factor 1). If α is a constant, such an algorithm is also referred to as a *constant factor approximation algorithm*. For an introduction into approximation algorithms and their design, we refer the reader to “The Design of Approximation Algorithms” by Williamson and Shmoys [122].

INAPPROXIMABILITY AND APPROXIMATION SCHEMES When designing polynomial time approximation algorithms for some optimization problem Π , it is only natural to ask for the best possible approximation factor α that can be achieved by any such algorithm. One way to find such an α is to not only give an α -approximation algorithm for Π , but to also show that there cannot exist an algorithm with approximation factor lesser than α unless some common conjecture fails, i. e. unless $\mathcal{P} = \mathcal{NP}$. The latter is called the *inapproximability* of a problem Π . A well-known example is the inapproximability of the Traveling Salesman Problem, for which no polynomial time algorithm with approximation factor less than $1^{23}/122$ can exist, unless $\mathcal{P} = \mathcal{NP}$ [84].

If such inapproximability results however do not exist for some problem Π , the approximation factor α of some polynomial time approximation algorithm can, in theory, get arbitrarily close to 1, i. e. one should be able to find an $(1 + \varepsilon)$ -approximation algorithm for all $\varepsilon > 0$. Such algorithms do in fact exist, and are called *Approximation Schemes*.

Definition 4 (Approximation Schemes). An approximation scheme for an optimization problem Π is a family of algorithms $\{\mathcal{A}_\varepsilon\}$ such that, for any $\varepsilon > 0$, $\mathcal{A}_\varepsilon$ is a $(1 + \varepsilon)$ -approximation algorithm (if Π is a minimization problem) or a $(1 - \varepsilon)$ -approximation algorithm (if Π is a maximization problem). Let $f : \mathbb{N}_{\geq 0} \mapsto \mathbb{N}_{\geq 0}$ be some computable function.

- If the run time for all $\mathcal{A}_\varepsilon$ is in $|I|^{f(\varepsilon^{-1})}$, for all instances I of Π , $\{\mathcal{A}_\varepsilon\}$ is called a Polynomial-Time Approximation Scheme (PTAS).
- If the run time of a PTAS is in $f(\varepsilon^{-1}) \cdot |I|^{\mathcal{O}(1)}$, it is called an Efficient Polynomial-Time Approximation Scheme (EPTAS).
- If the run time of an EPTAS is in $(\varepsilon^{-1}|I|)^{\mathcal{O}(1)}$, it is called a Fully Polynomial-Time Approximation Scheme (FPTAS).

PTAS, EPTAS and FPTAS also refer to the respective complexity classes that contain all problems that allow PTAS, EPTAS and FPTAS algorithms. This definition implies that $\text{FPTAS} \subseteq \text{EPTAS} \subseteq \text{PTAS}$. Note that there cannot exist FPTASs for strongly $\mathcal{NP}$ -hard optimization problem, unless $\mathcal{P} = \mathcal{NP}$ [119]. Approximation schemes are inter alia highly relevant in the area of scheduling, where for example much effort has been put into finding EPTASs with gradually improved run times for $Q||C_{\max}$ [17, Table 1].

The attentive reader may have already noticed the multitude of similarities between the definitions for parameterized algorithms and approximation schemes. In fact, many approximation schemes are found by first designing an XP or FPT algorithm for an optimization problem Π and some parameter k , and then rounding the input for any $\varepsilon > 0$ in such a way that $k \in \mathcal{O}(\varepsilon^{-1})$, which directly results in a PTAS or EPTAS for Π , respectively. This is done for example by Jansen and Rohwedder [81] for the problem of makespan minimization on parallel identical machines. One may also use parameterization and approximation simultaneously to solve a problem, if using only either one does not lead to the desired result. This kind of combination most relevant to this work is the notion of an Efficient Parameterized Approximation Scheme (EPAS).

Definition 5 (EPAS). Let k be some parameter and $f : \mathbb{N}_{\geq 0} \mapsto \mathbb{N}_{\geq 0}$ some computable function. An EPAS for an optimization problem Π is a family of algorithms $\{\mathcal{A}_\varepsilon\}$ such that, for any $\varepsilon > 0$, $\mathcal{A}_\varepsilon$ is a $(1 + \varepsilon)$ -approximation algorithm (if Π is a minimization problem) or a $(1 - \varepsilon)$ -approximation algorithm (if Π is a maximization problem), and $\mathcal{A}_\varepsilon$ computes such a solution in time $f(k, \varepsilon) \cdot \text{poly}(|I|)$ for any instance I of Π .

Such algorithms can thus be seen as FPT EPTASs, parameterized in k . As Abbasi et al. [5] remark, these algorithms are not directly comparable to PTASs w. r. t. to their computational complexity.

DUAL APPROXIMATION A concept originally from scheduling theory, but now used for the design of approximation algorithms in many different areas of optimization problems, is that of *dual approximation* by Hochbaum and Shmoys [69]. This concept can be formally defined as follows.

Definition 6 (Dual Approximation Algorithm). Let I be an instance of an optimization problem Π . An algorithm $\mathcal{A}$ is called an α -*dual approximation algorithm*, if, for any $T \in \mathbb{N}_{\geq 0}$, $\mathcal{A}$ either computes a feasible solution to I with approximation factor αT , or correctly concludes that no feasible solution with objective value T exists. In the former case, we also say $\mathcal{A}$ *accepts* T , in the latter we say $\mathcal{A}$ *rejects* T .

This kind of approximation technique is quite similar to the approach of solving optimization problems via their corresponding decision problem, and particularly useful if the number of possible objective values of feasible solutions can be described as an integer interval $[\ell, u]$ for every instance I of a problem Π . If we can find some $T \in [\ell, u]$ that is accepted by the algorithm, such that either $T - 1$ (if Π is a minimization problem) or $T + 1$ (if Π is a maximization problem) is rejected, we can conclude the αT -approximation is, in fact, an α -approximation, as either $T - 1 < \text{OPT}_{\Pi}(I)$ in the former, or $T + 1 > \text{OPT}_{\Pi}(I)$ in the latter case. Such a T can be found in $\mathcal{O}(\log(u - \ell))$ time via binary search. Thus, if it holds that $(u - \ell) \in 2^{\mathcal{O}(|I|)}$, a polynomial time α -dual approximation algorithm can be used as a sub-routine to obtain a polynomial time α -approximation algorithm by the steps described above. When introducing the concept, Hochbaum and Shmoys [69] made successfully use of this idea to find the first PTAS for makespan minimization on identical parallel machines by giving a trivial yet appropriate interval $[\ell, u]$ with $(u - \ell) \in 2^{\mathcal{O}(|I|)}$, and rounding the inputs in terms of ε for each guess T .

1.2 OVERVIEW

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

1.2.1 Constant Factor Approximation Algorithms

COUPLED TASK SCHEDULING MINIMIZING THE SUM OF COMPLETION TIMES. In this chapter, we consider the coupled task scheduling problem with exact delay times on a single machine with the objective of minimizing the total completion time of the jobs. We provide constant-factor approximation algorithms for several variants of this problem that are known to be $\mathcal{NP}$ -hard, while also proving $\mathcal{NP}$ -hardness for two variants whose complexity was unknown before. Using these results, together with constant-factor approximations for

the makespan objective from the literature, we also introduce the first results on bi-objective approximation in the coupled task setting.

Chapter 3

PREEMPTIVE SCHEDULING WITH BATCH SETUP TIMES. In this chapter, we consider the $\mathcal{NP}$ -hard problem $P|\text{pmtn, setup} = s_i|C_{\max}$, the problem of scheduling n jobs, which are divided into c classes, on m identical parallel machines while allowing preemption. For each class i of the c classes, we are given a setup time s_i that is required to be scheduled whenever a machine switches from processing a job of one class to a job from another class. The goal is to find a schedule that minimizes the makespan. We give a $(4/3 + \epsilon)$ -approximate algorithm with run time in $\mathcal{O}(n^2 \log(1/\epsilon))$. For any $\epsilon < 1/6$, this improves upon the previously best known approximation ratio of $3/2$ for this problem.

1.2.2 Parameterized Algorithms and Approximation Schemes

Chapter 4

FPT ALGORITHMS FOR ENVY MINIMIZATION ON UNIFORM MACHINES In this chapter, we consider the $\mathcal{NP}$ -hard problem of scheduling n jobs on m uniform machines while minimizing C_{envy} , i. e. the difference between the makespan and minimum completion time, of a schedule. This problem is known as $Q||C_{\text{envy}}$, and we study it in $HM(m, n)$ encoding. We give a $p_{\max}^{\mathcal{O}(d^2)}$ $\text{poly}(\log n, \log m, \tau, d)$ time algorithm for $Q|HM(m, n)|C_{\text{envy}}$. This, to the best of our knowledge, is the first result showing that $Q||C_{\text{envy}}$ is FPT only in the parameter $p_{\max}$, in the high-multiplicity setting or otherwise. We also introduce a speed-up to improve the run time for solving the same problem in time $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$. This is the asymptotically best possible run time for these parameters under the ETH, matching the run time for solving $Q|HM(n)|C_{\max}$ and $Q|HM(n)|C_{\min}$ from the literature.

Chapter 5

ROBUST SCHEDULING ON UNIFORM MACHINES. In this chapter, we consider the $\mathcal{NP}$ -hard problem of scheduling n jobs on m uniform machines while minimizing the makespan ($Q||C_{\max}$) and maximizing the minimum completion time ($Q||C_{\min}$) in an online setting with migration of jobs. In this online setting, the jobs are inserted or deleted over time, and at each step, the goal is to compute a near-optimal solution while reassigning some jobs, such that the overall processing time of reassigned jobs, called migration, is bounded by some factor β times the processing time of the job added or removed. We propose EPTASs with an additional load error of $\mathcal{O}(\epsilon p_{\max})$ for both problems, with constant amortized migration factor β , where $p_{\max}$ is the maximum processing time in the instance over all steps. As an intermediate step, we obtain EPASs for both problems, $(1 + \epsilon)$ -competitive algorithms parameterized by $p_{\max}$ and d , with β bounded in a function of $p_{\max}$, d and ϵ . This is the first result in the direction of a polynomial time

approximation scheme in the field of online scheduling with bounded reassignment on uniform machines; before, such results were known for both objective function only on identical machines.

Part I

CONSTANT FACTOR APPROXIMATION ALGORITHMS

COUPLED TASK SCHEDULING MINIMIZING THE SUM OF COMPLETION TIMES

2.1 INTRODUCTION

The problem of scheduling coupled tasks with exact delays, also known as Coupled Task Processing (CTP), was introduced by Shapiro [115] more than forty years ago. In this particular scheduling problem, each job has two separate tasks and a delay time. The goal is to schedule these tasks such that no tasks overlap, and the two tasks of a job are scheduled with exactly their given delay time in between them, while optimizing some objective function. This problem has several practical applications, e.g., in pulsed radar systems, where one needs to receive the reflections of the transmitted pulses after a given period of time [48, 50], in improving the performance of submarine torpedoes [116], or in chemistry [6].

Research interest in the coupled task problem is strongly increasing in recent years, see Khatami et al. [88] for a current, detailed overview of the topic. This research focuses mainly on variants of the general CTP with some additional restrictions on the job properties, and mainly tries to optimize the makespan [88]. Coupled task problems are often $\mathcal{NP}$ -hard even in very special cases, but polynomial-time approximation algorithms with constant approximation factors have been developed for a number of them [6, 7, 8]. Other objective functions have virtually not been considered though, until recently, when Chen and Zhang [29] drew an almost full complexity picture for the problems of single-machine scheduling of coupled tasks with the objective of minimizing the total sum of job completion times. However, they did not give any approximation algorithms for $\mathcal{NP}$ -hard CTP variants with this particular objective function. We fill this gap by giving a number of constant-factor approximation algorithms for most of these CTP variants. Additionally, we introduce two new, interesting variants, which we also prove to be $\mathcal{NP}$ -hard, and also approximate one of these with a constant factor.

Formally, we are given a set of n jobs $\mathcal{J} = \{1, 2, \dots, n\}$, where each job j has two tasks: a_j and b_j . We call a_j the first task, and b_j the second task of job j . In order to simplify our notations, we will denote the processing time of these tasks also by a_j and b_j ; the meaning of these notations will be clear from context. The sum $(a_j + b_j)$ is then called the total processing time of a job j . These tasks have to be scheduled on a single machine with a given delay time L_j in-between, which means if the machine completes a_j at some time point t , then we have to

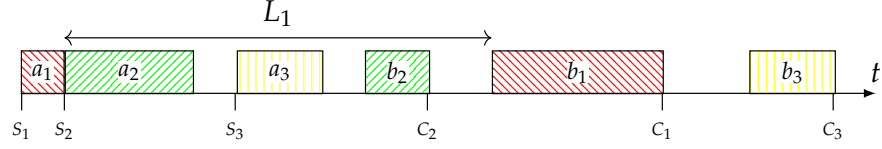


Figure 2.1: An example for a feasible solution for an instance of CTP with $n = 3$. The patterns are matching for the two tasks of each job j . For simplicity, the delay time is only visualized for job 1. The total completion time of the solution is $C_1 + C_2 + C_3$.

schedule b_j to start exactly at $t + L_j$. Preemption is not allowed. Note that it is possible to schedule other tasks on the machine during this delay time, but the tasks themselves cannot overlap. Our objective is to find a feasible schedule σ that minimizes the total of job completion times, where a feasible schedule is defined as a schedule that fulfills all of the requirements above. Such a σ is then called optimal schedule or optimal solution for the CTP instance. For a given schedule σ , the starting time S_j of j is the starting time of a_j , while the completion time C_j of j is the completion time of b_j . An example of CTP is visualized in Figure 2.1. For a schedule σ , a *gap* is a period between time points t_1 and t_2 such that the machine is idle between t_1 and t_2 and busy at both t_1 and t_2 . The length of a gap is the length of this time window. A partial schedule σ^p is a schedule for a subset of the jobs $\mathcal{J}$.

We say a job is the j^{th} finishing job in a schedule if its second task is scheduled after the second tasks of exactly $j - 1$ many jobs have finished. We say a job is the j^{th} starting job in a schedule if its first task is scheduled after the first tasks of exactly $j - 1$ many jobs have been scheduled. Let C^{OPT} be the sum of completion times of an optimal solution. For a fixed optimal schedule, let $C_j^{\text{OPT}_f}$ be the completion time of the j^{th} finishing job in that schedule. Analogously, let $C_j^{\text{OPT}_s}$ be the completion time of the j^{th} starting job in that schedule. Observe that $C^{\text{OPT}} = \sum_{j=1}^n C_j^{\text{OPT}_f} = \sum_{j=1}^n C_j^{\text{OPT}_s}$. In any proof in this chapter, we denote the sum of completion times of the solution produced by the currently used algorithm as C^{ALG} . In the schedule σ^{ALG} created by this algorithm, S_j and C_j denote the starting time and completion time of job j , respectively.

$1|(a_j, L_j, b_j)| \sum C_j$ denotes the general CTP for minimizing the sum of completion times, where each job j consists of a pair of tasks of processing times a_j and b_j , respectively, with an exact time delay L_j between the completion time of its first task and the start time of its second task. As we also look at more restricted variants of CTP, we fix some naming conventions to easily express these restrictions in Graham notation [57]. If in a restricted CTP environment, some tasks of the jobs are fixed or even constant for each job j , we denote this task without the subscript ' j ', or by the specific constant value; e.g. CTP, where the delay time L_j is fixed to some L for all jobs j is denoted as

$1|(a_j, L, b_j)| \sum C_j$. If in a restricted CTP environment, some tasks of the same job always have the same value, we denote them by p instead of their usual descriptor; e.g., CTP where the first and second task of each job j have the same processing time ($a_j = b_j, \forall j$) is denoted as $1|(p_j, L_j, p_j)| \sum C_j$. This is in line with the standard notation for the coupled task scheduling problems, as seen for example in Chen and Zhang [29]. Another way to restrict CTP is to fix the processing sequence of the first tasks of the jobs, which is indicated by π_a in the β -field of the Graham notation.

2.1.1 Related Work

Research on coupled task scheduling on a single machine began when Shapiro [115] proved the $\mathcal{NP}$ -hardness of the general problem $1|(a_j, L_j, b_j)| C_{\max}$, where both tasks, as well as the delay time between them, can be different for each job, and the makespan is to be minimized. In subsequent years, the $\mathcal{NP}$ -hardness was also shown for more restricted variants of this problem, specifically $1|(p_j, p_j, p_j)| C_{\max}$, $1|(a_j, L, b)| C_{\max}$, $1|(a, L, b_j)| C_{\max}$ and $1|(p, L_j, p)| C_{\max}$ by Orman and Potts [107]. Some CTP variants minimizing the makespan are $\mathcal{NP}$ -hard even when the processing times of all jobs are fixed to 1, as shown by Yu et al. [125] for $1|(1, L_j, 1)| C_{\max}$. Condotta and Shakhlevich [33] showed $\mathcal{NP}$ -hardness for the even more restricted variant $1|(1, L_j, 1, \pi_a)| C_{\max}$, where π_a indicates a fixed processing sequence for the first tasks of all jobs.

For most of these problems, polynomial-time constant-factor approximation algorithms have been developed. Ageev and Kononov [8] give such algorithms, as well as inapproximability bounds, for the general $1|(a_j, L_j, b_j)| C_{\max}$ problem, and the restricted variants $1|(a_j, L_j, b_j, a_j \leq b_j)| C_{\max}$, $1|(a_j, L_j, b_j, a_j \geq b_j)| C_{\max}$, and $1|(p_j, L_j, p_j)| C_{\max}$. Related to this work, Ageev and Baburin [6] give an approximation algorithm for the $1|(1, L_j, 1)| C_{\max}$ variant. Additionally, Ageev and Ivanov [7] give approximation algorithms and inapproximability bounds for $1|(a_j, L, b_j)| C_{\max}$, $1|(a_j, L, b_j, a_j \leq b_j)| C_{\max}$ and $1|(p_j, L, p_j)| C_{\max}$. These approximability results for the $C_{\max}$ -objective are summarized in Table 2.1. For other restricted variants, polynomial-time algorithms do exist. This was shown by Orman and Potts [107] for the variants $1|(p, p, b_j)| C_{\max}$, $1|(a_j, p, p)| C_{\max}$, and $1|(p, L, p)| C_{\max}$, as well as Hwang and Lin [72] for the variant $1|(p_j, p_j, p_j), \text{fjs}| C_{\max}$, where “fjs” denotes that the sequence of jobs in the schedule is fixed.

Research interest in the topic of coupled task scheduling remained high also in the last years. Békési et al. [27] recently introduced and gave a constant-factor approximation algorithm for the novel problem variant $1|(1, L_j, 1), L_j \in \{L_1, L_2\}| C_{\max}$, where there are only two different delay times in an instance; the complexity status of this variant is still unknown. Khatami and Selhipour [86] tackle the coupled task

Table 2.1: A summary of the approximability results for the $C_{\max}$ objective.

problem	appr. factor	inappr. bound	ref.
$1 (a_j, L_j, b_j) C_{\max}$	3.5	$2 - \varepsilon$	[8]
$1 (a_j, L_j, b_j, a_j \leq b_j) C_{\max}$	3	$2 - \varepsilon$	[8]
$1 (a_j, L_j, b_j, a_j \geq b_j) C_{\max}$	3	$2 - \varepsilon$	[8]
$1 (p_j, L_j, p_j) C_{\max}$	2.5	$2 - \varepsilon$	[8]
$1 (1, L_j, 1) C_{\max}$	1.75		[6]
$1 (a_j, L, b_j) C_{\max}$	3	$1.25 - \varepsilon$	[7]
$1 (a_j, L, b_j, a_j \leq b_j) C_{\max}$	2	$1.25 - \varepsilon$	[7]
$1 (p_j, L, p_j) C_{\max}$	1.5	$1.25 - \varepsilon$	[7]

scheduling problem differently, giving upper and lower bounds on the solution through different procedures, and proposing a binary heuristic search algorithm for CTP. The same authors give optimal solutions under certain conditions, and a general heuristic for the problem variant with fixed first tasks and delay times, but time-dependent processing times for the second tasks [87]. Bessy and Giroudeau [23] investigate CTP under parameterized complexity, with the considered parameter k relating to the question if k coupled tasks have a completion time before a fixed due date. Hadda and Abdesslem [62] show some structural results regarding the optimality of solutions for CTP problems. Kong et al. [93] consider a generalization of CTP, where each job may be rejected instead of scheduled, for some fixed rejection penalty.

Interest is also high in scheduling coupled tasks in *2-machine flow shop* environments, denoted by F2 in the machine environment notation. For scheduling coupled tasks in this environment, we are given two machines instead of one, and each of the two task of one job is additionally assigned one of these two machines to be processed on. $\mathcal{NP}$ -hardness is shown for a number of flow shop problems minimizing the makespan, e.g., for $F2|(1, L_j, 1)|C_{\max}$ by Yu et al. [125], but $\mathcal{NP}$ -hardness is also known for variants minimizing the total completion time, e.g., $F2|(a_j, L, b_j)|\sum C_j$, as shown by Leung et al. [97]. Several flow shop problem variants minimizing the total completion time are also polynomial solvable, as proven by Leung et al. [97] and Huo et al. [71]. Very recently, Khatami, Oron and Salehipour [85] started considering CTP problems on identical parallel machines, proving $\mathcal{NP}$ -hardness and inapproximability bounds for several variants, as well as giving polynomial time algorithms for variants in $\mathcal{P}$.

All of the mentioned literature for scheduling coupled tasks on a single machine only considers the objective of minimizing the makespan though, and, as Khatami et al. [88] note in their survey of CTP, “there has been no published research investigating the single-machine set-

ting with an objective function other than the makespan, except for those in the cyclic setting.” This task is finally tackled by Chen and Zhang [29], who draw a nearly full complexity picture of problem of minimizing the total of job completion times. However, they do not give any approximation algorithms for problem variants they prove to be $\mathcal{NP}$ -hard. Recently, Kubiak [95] slightly extended these complexity results by proving $\mathcal{NP}$ -hardness of $1|(1, L_j, 1)| \sum C_j$ and $1|(1, L_j, 1)| \sum C_j$. In the latter problem variant, the delay time between the two tasks does not have to be exactly, but at most L_j .

In scheduling theory, there is also a great interest in bi-objective and multi-objective optimization. Here, instead of trying to optimize just one objective function in a given problem setting, one aims to optimize two or more objective functions at the same time, see Deb [40] or Hoogeveen [70] for an overview. Since, until recently, virtually only the makespan objective has been considered for coupled tasks scheduling problems, we do not know of any such results in the couple task environment. This is not true for other scheduling environments though, where especially bi-objective optimization is intensively researched, particularly for the two objectives of minimizing makespan and sum of completion times. Here, many approaches focus on establishing a trade-off relationship between the two competing objectives, either by *Pareto optimization* (finding one or all Pareto optimal solutions) or *simultaneous optimization* (minimizing all objectives without prioritization) [82]. Since these problems are generally $\mathcal{NP}$ -hard (see e.g. Hoogeveen [70]), approximation is a popular method for both mentioned approaches. Angel et al. [11] give fully polynomial time approximation schemes for the Pareto curve of single-machine batching problems and parallel machine scheduling problems on the two objectives. Bampis and Knonov [14] consider (ρ_1, ρ_2) -approximations of the two objectives for scheduling problems with communication delays. A very recent work by Jiang et al. [82] is concerned with (ρ_1, ρ_2) -approximations for scheduling on parallel machines, with different approximation ratios for different fixed numbers of machines.

2.1.2 Our Contribution

We extend the complexity results of Chen and Zhang [29] and Kubiak [95] by proving the strong $\mathcal{NP}$ -hardness of $1|(p_j, L, p_j)| \sum C_j$ and $1|(1, L_j, 1, \pi_a)| \sum C_j$. To achieve the former, we first prove strong $\mathcal{NP}$ -hardness of the corresponding makespan variant $1|(p_j, L, p_j)| C_{\max}$, strengthening a result by Ageev and Ivanov [7], who prove only weak $\mathcal{NP}$ -hardness of this problem. We also give constant-factor approximations for most CTP variants in a single machine environment with the sum of completion times objective function, see Figure 2.2. The existence of a constant-factor approximation algorithm for the variants

$1|(a_j, L_j, b_j)|\sum C_j$, $1|(a, L_j, b_j)|\sum C_j$, $1|(a_j, L_j, b)|\sum C_j$, and $1|(p_j, L_j, p_j)|\sum C_j$ remains open (see the upper part of the figure).

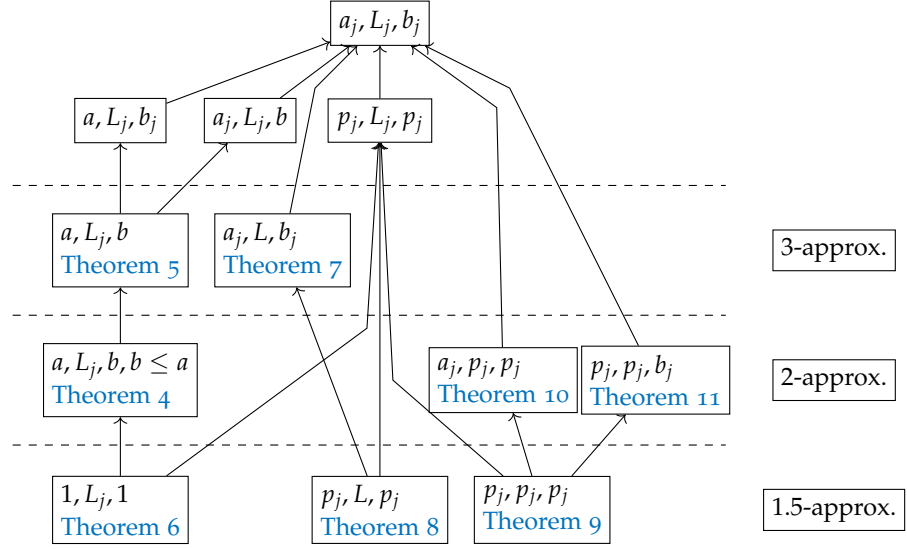


Figure 2.2: Overview of our approximation results for different variants of $1|(a_j, L_j, b_j)|\sum C_j$. The variants, identified by their special constraints, are grouped into layers of equal approximation factors, with the respective Theorem proving this approximation factor linked next to it. A directed edge from variant “A” to variant “B” indicates that “B” is a generalization of “A”.

We also look at bi-objective optimization for CTP with both the makespan and the sum of completion times objectives, under the goal of minimizing both objectives without prioritization. For this, we use the concept of $(\rho_1, \dots, \rho_z)$ -approximation, as introduced by Jiang et al. [82] for simultaneously minimizing z objectives, where ρ_i is the approximation factor of the i^{th} objective function to be minimized, for $i = 1, \dots, z$. This concept is a generalization of the bi-objective (ρ_1, ρ_2) -approximation of scheduling problems minimizing makespan and sum of completion times, as described first by Stein and Wein [118]. As far as we are aware, there are no results of minimizing the two objectives makespan and sum of completion times simultaneously in a coupled task setting, even though this topic is well researched in other scheduling environments. We start to close this gap by directly using results of Stein and Wein [118], together with constant-factor approximations of the makespan [6, 7, 8] and our approximation results on the sum of completion times objective, to give a number of (ρ_1, ρ_2) -approximation results for this problem. The general bi-objective CTP is denoted as $1|(a_j, L_j, b_j)|(C_{\max}, \sum C_j)$ in the $\alpha|\beta|\gamma$ notation system, with the β -field following the previously discussed naming conventions depending on the considered variant’s restrictions.

ORGANIZATION OF THIS CHAPTER We start by presenting our complexity results in Section 2.2. Our approximation results are stated

in Section 2.3. There, we give detailed descriptions and run time analyses of our algorithms, as well as proofs of the approximation factors for problem variants whose instances can be solved by these algorithms. We then use these results in Section 2.4 to give (ρ_1, ρ_2) -approximations for the bi-objective CTP with both the makespan and sum of completion time objectives. Finally, we give concluding remarks and an outlook on future research in Section 2.5.

2.2 COMPLEXITY RESULTS

In this section we prove that both $1|(p_j, L, p_j)| \sum C_j$ and $1|(1, L_j, 1, \pi_a)| \sum C_j$ are strongly $\mathcal{NP}$ -hard. We use reductions from the corresponding makespan minimization problems for both results. As we need strong $\mathcal{NP}$ -hardness of the corresponding problems for both our reductions, we additionally prove strong $\mathcal{NP}$ -hardness of $1|(p_j, L, p_j)| C_{\max}$; weak $\mathcal{NP}$ -hardness was already proven for this problem by Ageev and Ivanov [7].

Theorem 1. $1|(p_j, L, p_j)| C_{\max}$ is strongly $\mathcal{NP}$ -hard.

Proof. We reduce the well-known strongly $\mathcal{NP}$ -hard problem 3-PARTITION [55] to $1|(p_j, L, p_j)| C_{\max}$. The reduction is similar to the idea by Ageev and Ivanov [7] for reducing the weakly $\mathcal{NP}$ -hard PARTITION problem to $1|(p_j, L, p_j)| C_{\max}$. First, let us formally state the 3-PARTITION problem.

3-PARTITION

Instance: A set $Q = \{1, \dots, 3q\}$, and for each element $i \in Q$, a corresponding positive integer e_i such that $\sum_{i \in Q} e_i = qE$, for some positive integer E , and $E/4 < e_i < E/2$.

Question: Does the set Q partition into q disjoint subsets $Q_1, \dots, Q_q$ such that $\sum_{i \in Q_j} e_i = E$, for $j = 1, \dots, q$?

Consider an instance I of 3-PARTITION. We define an instance I' of $1|(p_j, L, p_j)| C_{\max}$ with $4q$ jobs as follows:

- jobs $i = 1, \dots, 3q$ have $p_i = e_i$ and $L = R + E$ (small jobs),
- jobs $i = 3q + 1, \dots, 4q$ have $p_i = R$ and $L = R + E$ (large jobs),

for some $R > 3qE$. If q is odd, we additionally introduce a dummy job i with $p_i = E$ and $L = R + E$. We prove there is a solution for I if and only if there is a solution for I' with makespan $C_{\max} \leq z$, with $z := q(3E + 2R)$. The theorem follows from this statement.

1. Assume there is a solution for I . Then there exist $Q_1, \dots, Q_q$, such that $\sum_{i \in Q_j} e_i = E$, for $j = 1, \dots, q$. In this case, we create a schedule σ for I' with makespan at most z . We schedule the jobs

in blocks. Let B be an arbitrary block. It consists of 2 large jobs (i and i') and 6 small jobs, which corresponds to items from some Q_j and $Q_{j'}$. If q is odd, assume that one such block contains 3 small jobs and the dummy job with task time E . Let (j_1, j_2, j_3) and (j'_1, j'_2, j'_3) be the small jobs corresponding to the items in Q_j and $Q_{j'}$, respectively. Assume that $p_{j_1} \geq p_{j_2} \geq p_{j_3}$ and $p_{j'_1} \leq p_{j'_2} \leq p_{j'_3}$. We schedule $a_{i'}$ directly before b_i . We schedule b_{j_1}, b_{j_2} and b_{j_3} in this order in the gap between a_i and $a_{i'}$ and $a_{j'_1}, a_{j'_2}$ and $a_{j'_3}$ in this order in the gap between b_i and $b_{i'}$, see Figure 2.3. Observe that the job tasks do not intersect, because the length of the gap between a_i and $a_{i'}$ as well as the gap between b_i and $b_{i'}$ is exactly E . The length of block B is at most $4R + 6E$, because a_{j_1} starts at most $2E$ before a_i , and b_{j_3} completes at most $2E$ after $b_{i'}$. We create such blocks for all jobs, resulting in $q/2$ blocks in total, and schedule these blocks directly one after another. Thus, the resulting schedule σ has makespan $C_{\max} \leq q/2(4R + 6E) = z$.

2. Now assume that the instance I' has a schedule σ with makespan $C_{\max} \leq z$. Due to the fixed delay times, the order is the same for the first and the second tasks in σ . Consider an arbitrary large job i . There has to be exactly one task of another large job i' between a_i and b_i . There cannot be more than one task of a large job between a_i and b_i . If there was no task of a large job between a_i and b_i , the makespan of σ would be larger than z , as the total processing time of the large jobs is $2qR$, and $L = R + E$, resulting in a minimum makespan of $C_{\max} \geq 2qR + (R + E) > q(2R + 3E) = z$, due to $R > 3qE$. Observe that if a task of job i' is scheduled in the delay time of i , then a task of i is scheduled in the delay time of i' . This means we can partition the large jobs into pairs where the jobs within each pair are interleaved in the above way.

Consider an arbitrary pair of large jobs (i, i') and assume that $S_i < S_{i'}$. There cannot be any task of a small job scheduled between $a_{i'}$ and b_i , as a first task of a small job would imply an intersection of its second task with $b_{i'}$, and a second task of a small job would imply an intersection of its first task with a_i , due to the fixed delay times. Consider an arbitrary small job j . If none of its tasks is scheduled in the gap between a_i and $b_{i'}$ of any pair of large jobs (i, i') , then there is also no task of a large job scheduled in the delay time of j due to the fixed delay times. This implies the makespan of σ is at least $2qR + 2e_j + R + E > z$, because the total processing time of the large jobs together with j is $2qR + 2e_j$, and the delay time of j is $R + E$.

Therefore, for any small job j , there is a pair of large jobs (i, i') , where at least one of the tasks of j is scheduled in the gap between a_i and $a_{i'}$ or in the gap between b_i and $b_{i'}$. For any pair

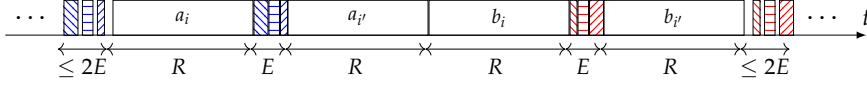


Figure 2.3: A block B of jobs in σ . The blue tasks are j_1, j_2 and j_3 , while the red tasks are j'_1, j'_2 and j'_3 (in this order).

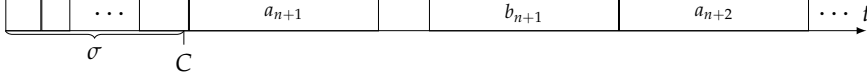


Figure 2.4: Schedule σ' created from schedule σ

of large jobs the length of the gap between a_i and $a_{i'}$ is at most E , the same holds for the length of the gap between b_i and $b_{i'}$. Thus, the total length of these gaps is at most $q/2 \cdot (2E) = qE$, which is exactly half of the total processing time of the small jobs. Therefore, the length of each such gap must be exactly E , and they must be completely filled with tasks of small jobs. As $E/4 < e_j < E/2$, there are always exactly three tasks of small jobs in each such gap. This partitions the small jobs into q sets Q_j of 3 jobs each, with $\sum_{x \in Q_j} e_x = E$ for each $j = 1, \dots, q$, which gives us a feasible solution for the 3-PARTITION instance I .

□

Theorem 2. $1|(p_j, L, p_j)| \sum C_j$ is strongly $\mathcal{NP}$ -hard.

Proof. We reduce the strongly $\mathcal{NP}$ -hard problem $1|(p_j, L, p_j)| C_{\max}$, proven to be strongly $\mathcal{NP}$ -hard in [Theorem 1](#), to $1|(p_j, L, p_j)| \sum C_j$. Consider an instance I of $1|(p_j, L, p_j)| C_{\max}$. We define an instance I' of $1|(p_j, L, p_j)| \sum C_j$ as follows. There are $n + M$ jobs in I' , where M is a sufficiently large number.

- the first n jobs are the same as the jobs in I (small jobs),
- for the remaining M jobs, we have $p_j = \sum_{i=1}^n p_i + nL$, $j = n + 1, \dots, n + M$ (large jobs).

We prove that there is a solution σ for I with makespan at most C if and only if there is a solution for I' with a total completion time of at most $z := (n + M)C + hM(M + 1)/2$, where $h := 2(\sum_{i=1}^n p_i + nL) + L = a_j + L + b_j$ is the time required for scheduling a large job j . The Theorem follows from this statement.

If there is such an σ , then we define σ' (a solution of I') from σ as follows. The small jobs start exactly at the same time as they start in σ , while the large jobs start right after them as soon as possible (in arbitrary order), see [Figure 2.4](#). Observe that the completion time of any small job in σ' is at most C , while the total completion time of the large jobs is $MC + hM(M + 1)/2$. Therefore, the total completion time of σ' is at most z .

Now suppose that there is no such solution for I , i.e. the makespan $C_{\max}$ of any solution σ is at least $C + 1$. Consider an arbitrary optimal solution $\hat{\sigma}$ for I' . Observe that no job task can be scheduled between the first and the second task of any large job in $\hat{\sigma}$, since the processing times of both tasks of any large job are larger than L . Furthermore, no large job can precede any of the small jobs, otherwise, we would get a better schedule by moving that large job to the end of the schedule (cf., the definition of the processing times of the large jobs). Therefore, the large jobs start right after the small jobs, i.e., after $C_{\max}$. Hence,

$$\begin{aligned} \sum_{j=1}^{n+M} \hat{C}_j &= \sum_{j=1}^n \hat{C}_j + MC_{\max} + hM(M+1)/2 \\ &> M(C+1) + hM(M+1)/2 > z, \end{aligned}$$

where the last inequality follows if M is larger than nC . $\square$

Theorem 3. $1|(1, L_j, 1, \pi_a)| \sum C_j$ is strongly $\mathcal{NP}$ -hard.

Proof. We reduce the known strongly $\mathcal{NP}$ -hard problem $1|(1, L_j, 1, \pi_a)|C_{\max}$, proven to be strongly $\mathcal{NP}$ -hard by Condotta and Shakhlevich [33], to $1|(1, L_j, 1, \pi_a)| \sum C_j$. Recall that π_a fixes a scheduling sequence for the first tasks of all jobs. Consider an instance I of $1|(1, L_j, 1, \pi_a)|C_{\max}$. We want to know if I has a solution with $C_{\max} \leq C$. We define an instance I' of $1|(1, L_j, 1, \pi_a)| \sum C_j$ as follows. There are $n + M$ jobs in I' , where M is a sufficiently large number.

- the first n jobs are the same as the jobs in I (original jobs),
- for the remaining M jobs, we have $L_j = C + 2(j - n - 1)$, $j = n + 1, \dots, n + M$ (helper jobs).

The fixed sequence of the first tasks in I' is defined as π'_a , with

- $\pi'_a := (n + M, n + M - 1, \dots, n + 1, \{\pi_a\})$.

We prove that there is a solution σ for I with makespan at most C if and only if there is a solution for I' with a total completion time of at most

$$z := n(M + C) + M^2 + \frac{M(M+1)}{2} + MC.$$

If there is such an σ , then we define σ' (a solution of I') from σ as follows. The helper jobs are scheduled in decreasing order of their indices one after another as soon as possible, while the original jobs are scheduled in the exact same way as in σ , but with their starting time increased by M each. See the first schedule of Figure 2.5 for an illustration. This schedule respects the order given by π'_a . Observe that the completion time of any original job in σ' is at most $M + C$, while the total completion time of the helper jobs is exactly $M^2 + \frac{M(M+1)}{2} + MC$. Therefore, the total completion time of σ' is at most z .

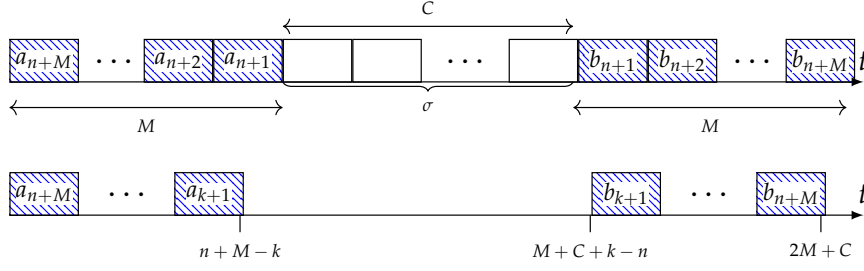


Figure 2.5: Above: schedule σ' created from schedule σ . The original jobs are white, the helper jobs are blue. Below: jobs $k+1, \dots, n+M$ in schedule $\hat{\sigma}$.

Now suppose that there is no such solution for I , i.e. the makespan $C_{\max}$ of any solution σ is at least $C+1$. Consider an arbitrary optimal solution $\hat{\sigma}$ for I' . Suppose for the sake of contradiction that the total completion time of $\hat{\sigma}$ is at most z for any M .

Due to the fixed order of the first tasks π'_a we can assume that a_{n+M} starts at 0 in $\hat{\sigma}$. Observe that the total completion time of the helper jobs is at least $M^2 + \frac{M(M+1)}{2} + MC$. The original jobs start after the first tasks of the helper jobs, i.e. after M , thus their total completion time is at least nM . If any of the original jobs completes after b_{n+M} in $\hat{\sigma}$, then its completion time is larger than $2M$, and the total completion time of the jobs in $\hat{\sigma}$ is larger than $M^2 + \frac{M(M+1)}{2} + MC + (n+1)M > z$, if $M > nC$. Thus, each original job has to start before b_{n+M} .

If there is no gap among the first tasks of the helper jobs in $\hat{\sigma}$, then the machine is busy with the helper jobs in $[0, M]$ and in $[M+C, 2M+C]$. Since each original job has to be completed before b_{n+M} (i.e. before $2M+C$), these jobs have to be scheduled in $[M, M+C]$. However, this is a contradiction to the the makespan of σ .

Therefore, in the following we suppose there is a gap between some first tasks of the helper jobs in $\hat{\sigma}$. Let k , $n+1 \leq k \leq n+M-1$, be the largest index such that there is a gap before a_k . Then, the machine is busy with jobs $n+M, n+M-1, \dots, k+1$ in $[0, n+M-k]$ and in $[M+C+k-n, 2M+C]$, see the second schedule in Figure 2.5. Since a_k starts later than $n+M-k$, b_k starts later than $M+C+k-n$, which means it starts later than $2M+C$ to avoid intersection.

Let j be a job in $\{n+2, \dots, k\}$ whose second task b_j is scheduled after b_{n+M} . Since $L_{j-1} = L_j - 2$, and a_{j-1} has to be scheduled after a_j according to π'_a , b_{j-1} is scheduled either immediately before, or at some time after b_j . In both cases, b_{j-1} has to be scheduled after b_{n+M} to avoid intersection with b_{n+M} . This particularly implies that b_{n+1} is scheduled after b_{n+M} , i.e. $C_{n+1} \geq C_{n+M} + 1 = 2M + C + 1$.

The total completion time of jobs $n+2, \dots, n+M$ is then at least $(M+C+2) + (M+C+3) + \dots + (2M+C) = (M-1)(M+C+2) + (M-1)(M-2)/2$. The total completion time of the original jobs is

at least nM . Thus we have the following lower bound on the total completion time of $\hat{\sigma}$:

$$\begin{aligned} & (2M + C + 1) + \left((M - 1)(M + C + 2) + \frac{(M - 1)(M - 2)}{2} \right) + (nM) \\ &= M^2 + M + CM + \frac{M^2 + M}{2} + nM > z, \end{aligned}$$

if $M > nC$, proving the Theorem. $\square$

2.3 APPROXIMATION RESULTS

As in the following, we only look at CTP with the objective of minimizing the total completion times, we call this problem only “CTP” from now on for simplicity reasons. In this section we give polynomial-time approximation algorithms for a number of CTP variants. All of these variants are proven to be $\mathcal{NP}$ -hard either by Chen and Zhang [29], by Kubiak [95], or by the results of Section 2.2. We start this section with two useful lemmas that provide lower bounds on the objective value of any optimal solution for the general CTP. Section 2.3.2 and Section 2.3.3 consider variants with fixed processing times and fixed delay times, respectively. Section 2.3.4 examines variants where there exists some relation between the processing times and the delay time of each job.

2.3.1 Lower Bounds on the Optimum

Recall the definitions $C_j^{\text{OPT}_f}$ and $C_j^{\text{OPT}_s}$ as the completion time of the j^{th} finishing and j^{th} starting job of some optimal schedule, respectively. The next lemma is straightforward from the definition of $C_j^{\text{OPT}_f}$, since there are j jobs that complete until $C_j^{\text{OPT}_f}$ in an optimal schedule.

Lemma 2. *Let the jobs of a CTP instance be indexed in non-decreasing $a_i + b_i$ order. Then, for any optimal schedule for this instance, we have*

1. $C_j^{\text{OPT}_f} \geq \sum_{i=1}^j (a_i + b_i)$, $j = 1, \dots, n$ and
2. $C^{\text{OPT}} \geq \sum_{j=1}^n \sum_{i=1}^j (a_i + b_i)$.

The second lemma is analogous, and follows from the observation that there are $j - 1$ first tasks that finish until the first task of the job corresponding to $C_j^{\text{OPT}_s}$ starts in some optimal schedule. See Figure 2.6 for an illustration.

Lemma 3. *Let the jobs of a CTP instance be indexed in non-decreasing a_i order. Then, for any optimal schedule for this instance, where j' is the j^{th} starting job, we have*

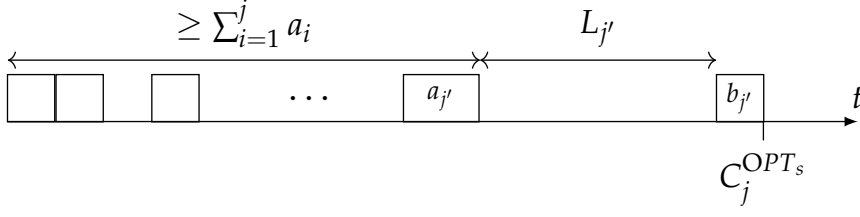


Figure 2.6: Illustration of the bound on $C_j^{OPT_s}$.

1. $C_j^{OPT_s} \geq \sum_{i=1}^j a_i + L_{j'} + b_{j'}, j = 1, \dots, n$ and
2. $C^{OPT} \geq \sum_{j=1}^n \sum_{i=1}^j a_i + \sum_{j=1}^n L_j + \sum_{j=1}^n b_j$.

2.3.2 CTP with Fixed Processing Times

In this section we assume $a_j = a$ and $b_j = b$ for each job $j \in \mathcal{J}$. Consider [Algorithm 1](#).

Algorithm 1

Input: a CTP instance with $a_j = a$ and $b_j = b$ for each job j

Output: a schedule σ for this instance

1. Sort the jobs in non-decreasing L_j order. In the following, let the jobs be indexed in this order.
 2. Schedule the jobs one-by-one as soon as possible without intersection.
-

Recall that here, σ^{ALG} denotes the schedule created by [Algorithm 1](#) and S_j is the starting time of job j . The precise meaning of ‘as soon as possible’ is the following: for a given (partial) schedule σ^p and a job j , scheduling j as soon as possible means setting its starting time $S_j := \min\{t \geq 0 : \text{the machine is idle both in } [t, t + a_j] \text{ and in } [t + a_j + L_j, t + a_j + L_j + b_j]\}$.

Lemma 4. *Suppose that $a_j = a$, $b_j = b$ ($j \in \mathcal{J}$) and $b \leq a$. When [Algorithm 1](#) schedules job j , where j is the j^{th} job in non-decreasing L_j order, then a_j starts from the earliest time point t' , such that the machine is idle in $[t', t' + a]$.*

Proof. We prove the statement by induction on j in the order given by [Step 1](#) in [Algorithm 1](#); it is trivial for $j = 1$. Suppose that the statement is true for some j , and consider the partial schedule σ^p created by the algorithm before scheduling job $j + 1$. Let t be the earliest time point such that the machine is either idle in $[t', t' + a]$ in σ^p (it is possible that t' is the time point when σ^p finishes). We prove the lemma by

showing $S_{j+1} = t$. Due to the definition of t' , a_{j+1} cannot intersect with any task in σ^p . We need to prove the same for b_{j+1} , which is scheduled in $[t' + a + L_{j+1}, t' + a + L_{j+1} + b]$.

First, observe that a_i starts before a_{i+1} for each $i \leq j$ due to the induction. This means $a_1, \dots, a_j$ complete before t' . Let $L_1, \dots, L_n$ be the delay times of the jobs obtained in [Step 1](#) of [Algorithm 1](#). Since $L_1, \dots, L_j \leq L_{j+1}$, we have $C_1, \dots, C_j \leq t' + L_{j+1} + b \leq t' + L_{j+1} + a$, where the last inequality follows from $b \leq a$. Thus, as each task in σ^p completes before the start of b_{j+1} , there is no intersection of any scheduled tasks. $\square$

Lemma 5. [Algorithm 1](#) runs in $\mathcal{O}(n \log n)$ time and it always produces a feasible solution.

Proof. Sorting the jobs requires $\mathcal{O}(n \log n)$ time. When [Algorithm 1](#) schedules job j , it searches the first gap after the first task of the directly previous scheduled job with a length of at least a ([Lemma 4](#)). This means the length of the gap after each task is only checked once during the whole procedure, which requires $\mathcal{O}(n)$ time in total. The feasibility of the schedule is straightforward from the definition of ‘as soon as possible’. $\square$

Theorem 4. [Algorithm 1](#) is a factor-2 approximation for $1|(a, L_j, b, b \leq a)| \sum C_j$.

Proof. Due to [Lemma 5](#), it remains to prove that $C^{\text{ALG}} \leq 2C^{\text{OPT}}$.

From [Lemma 4](#) we know that the machine is always busy just before a first task of a job is scheduled (except a_1 , which starts at time 0). This means there are at most $j - 1$ gaps before S_j , and the length of any of these gaps is smaller than a . Hence, if we consider the partial schedule at the time when the algorithm schedules job j , we have

$$S_j \leq (j - 1)(a + b) + (j - 1)a, \quad j \in \mathcal{J},$$

because (i) the machine is busy at most $(j - 1)(a + b)$ time in $[0, S_j]$ with tasks of the jobs $1, \dots, j - 1$; and (ii) the total length of the gaps before S_j is smaller than $(j - 1)a$. Therefore, we have $C_j \leq (j - 1)(a + b) + (j - 1)a + a + L_j + b$ and

$$\begin{aligned} C^{\text{ALG}} &= \sum_{j=1}^n C_j \leq \frac{n(n-1)}{2}(a+b) + \frac{n(n-1)}{2}a + n(a+b) + \sum_{j=1}^n L_j \\ &\leq C^{\text{OPT}} + \frac{n(n+1)}{2}a + nb + \sum_{j=1}^n L_j \leq 2C^{\text{OPT}}, \end{aligned}$$

where the second inequality follows from [Lemma 2](#) and from $n(n-1)/2 + n = n(n+1)/2$, while the third follows from [Lemma 3](#). $\square$

As $1|(a, L_j, b, b \leq a)| \sum C_j$ is a more general version of the variant $1|(p, L_j, p)| \sum C_j$, [Corollary 1](#) directly follows.

Corollary 1. *Algorithm 1 is a factor-2 approximation for $1|(p, L_j, p)| \sum C_j$.*

The following lemma describes some important attributes of σ^{ALG} in the opposite case of $a \leq b$.

Lemma 6. *Suppose that $a_j = a$, $b_j = b$ ($j \in \mathcal{J}$) and $a \leq b$. Consider the partial schedule σ^p created by Algorithm 1, right before it schedules job j starting at S_j . Then,*

- (i) a_j or b_j start right after another task (for each $j > 1$),
- (ii) the number of gaps before S_j is at most $j - 1$,
- (iii) the length of each gap in $[0, S_j]$ is at most b .

Proof. Statement (i) immediately follows from the algorithm. We use induction on j for proving the remaining statements (ii) and (iii); they are trivial for $j = 1$. Suppose that they are true for some $j \geq 1$, we then prove them for $j + 1$.

Consider (ii). If a_{j+1} or b_{j+1} either directly precede, or start right after some previously scheduled a_k , $k < j + 1$, in σ^p , the statement immediately follows. Otherwise, a_{j+1} or b_{j+1} either precede, or start right after some b_k with $k < j + 1$. In the latter, a_{j+1} either precedes, or starts right after a_k due to the non-decreasing delay time order, from which the statement also immediately follows. In the former (if a_{j+1} starts right after b_k), (ii) follows from the induction, since there is no gap in σ^p right before a_ℓ or b_ℓ for each $\ell \leq j$ (as known from (i)). Thus, there are at most j tasks before a_{j+1} which start right after a gap.

For proving (iii), suppose for purpose of showing contradiction that there exists a gap $[g_1, g_2]$ before b_k in σ^p with a length larger than b . We know from the induction that each a_ℓ ($\ell \leq j$) completes before g_1 . If $[g_1, g_2]$ existed however, the algorithm would schedule a_{j+1} at time g_1 : there would be neither an intersection between a_{j+1} and any other task (since $a \leq b < g_2 - g_1$), nor between b_{j+1} and any other task (since a_{j+1} starts later than the previously scheduled first tasks, and $L_{j+1} \geq L_\ell$ for each $\ell \leq j$). The existence of such a gap would contradict the definition of the algorithm, therefore, (iii) follows. $\square$

Theorem 5. *Algorithm 1 is a factor-3 approximation for $1|(a, L_j, b)| \sum C_j$.*

Proof. Due to Lemma 5, it remains to prove that $C^{\text{ALG}} \leq 3C^{\text{OPT}}$.

If $b \leq a$, we can approximate $1|(a, L_j, b)| \sum C_j$ with a factor of 2 (Theorem 4). Therefore, wlog., we assume in the following that $a \leq b$. We show that, in this case, we get an approximation factor of 3.

From Lemma 6, (ii) and (iii) we know that when the algorithm schedules job j , the total idle time of the machine in $[0, S_j]$ is at most $(j - 1)b$. Due to the order of the jobs, the machine is busy in the partial schedule from 0 to S_j for at most $(a + b)(j - 1)$ time; thus we have

$S_j \leq (a+b)(j-1) + b(j-1)$ and $C_j \leq (a+b)(j-1) + b(j-1) + (a+L_j+b)$. Hence,

$$\begin{aligned} C^{\text{ALG}} &= \sum_{j=1}^n C_j \leq \sum_{j=1}^n (a+b)(j-1) + \sum_{j=1}^n b(j-1) + n(a+b) + \sum_{j=1}^n L_j \\ &= n(n-1)b + \left(\frac{n(n+1)}{2}a + nb + \sum_{j=1}^n L_j \right) \leq n(n-1)b + C^{\text{OPT}}, \end{aligned}$$

where the second inequality follows from [Lemma 3](#). Since $C_j^{\text{OPT}_f} \geq (a+b)j$ by [Lemma 2](#), we have $C^{\text{OPT}} \geq \frac{n(n+1)}{2}(a+b)$. Therefore,

$$C^{\text{ALG}} \leq C^{\text{OPT}} + n(n-1)b \leq 3C^{\text{OPT}}. \quad \square$$

Theorem 6. *Algorithm 1 is a factor-1.5 approximation for $1|(1, L_j, 1)| \sum C_j$.*

Proof. Due to [Lemma 5](#), it remains to prove that $C^{\text{ALG}} \leq 1.5C^{\text{OPT}}$. From [Lemma 4](#) we know that the algorithm always schedules a_j in the first gap. Thus, the starting time of j ($j \in \mathcal{J}$) is at most $2(j-1)$, because the total processing time of the jobs $\ell < j$ is $2(j-1)$ and the other jobs start later. Therefore, $C_j \leq 2(j-1) + 2 + L_j$, and

$$\begin{aligned} C^{\text{ALG}} &\leq 2 \sum_{j=1}^n (j-1) + 2n + \sum_{j=1}^n L_j \\ &\leq C^{\text{OPT}} + \sum_{j=1}^n (j-1) + n \leq 1.5C^{\text{OPT}}, \end{aligned}$$

where the second inequality follows from [Lemma 3](#), and the third from [Lemma 2](#). $\square$

2.3.3 CTP with Fixed Delay Times

In this section we assume $L_j = L$ for each job $j \in \mathcal{J}$. Consider [Algorithm 2](#), based on an idea of Ageev and Ivanov [7]. Observe that both the first and the second tasks are in non-decreasing $a_j + b_j$ order in the schedule found by [Algorithm 2](#).

Lemma 7. *Algorithm 2 runs in $\mathcal{O}(n \log n)$ time and always produces a feasible solution.*

Proof. The run time of [Algorithm 2](#) is straightforward. Together with the observation that in [Step 3iii](#), we schedule the current job after all the other tasks, it is clear that there are no intersections in the schedule produced by the algorithm. $\square$

Let $B_s := \{j_s, j_s + 1, \dots, j_{s+1} - 1\}$ be the s^{th} block ($1 \leq s \leq n$) and let $H_s := C_{j_{s+1}-1} - S_{j_s}$ denote the length of B_s . The next lemma describes an important observation on the gap sizes within a block and follows directly from the equal delay times and some simple algebraic calculations (see [Figure 2.7](#) for illustration).

Algorithm 2**Input:** a CTP instance with $L_j = L$ for each job j **Output:** a schedule σ for this instance

1. Sort the jobs in non-decreasing $a_j + b_j$ order. In the following, let the jobs be indexed in this order.
2. Schedule a_1 from 0 and b_1 from $a_1 + L$. Let $s := 1$ and $j_s := 1$.
3. For $j = 2, \dots, n$ do the following:
 - i) If it is possible to start a_j immediately after a_{j-1} without b_j intersecting any job task on the schedule, then do it and consider the next job.
 - ii) If it is possible to start b_j right after b_{j-1} without a_j intersecting any job tasks on the schedule, then do it and go to consider the next job.
 - iii) Otherwise, start a_j immediately after b_{j-1} . Let $s := s + 1$ and $j_s := j$.

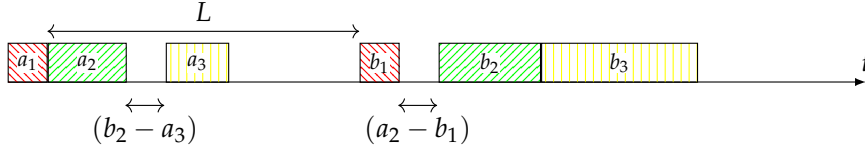


Figure 2.7: Example for gaps in different cases.

Lemma 8. *If jobs j and $j + 1$ are in the same block, then a_{j+1} starts immediately after a_j or b_{j+1} starts immediately after b_j . In the former case the gap between the second tasks is at most $a_{j+1} - b_j$, while in the latter case the gap between the first tasks is at most $b_j - a_{j+1}$.*

Theorem 7. *Algorithm 2 is a factor-3 approximation for $1|(a_j, L, b_j)| \sum C_j$.*

Proof. Due to Lemma 7, it remains to prove that $C^{\text{ALG}} \leq 3C^{\text{OPT}}$. The length of a block B_s is the sum of the following: (i) the lengths of the first tasks of the jobs in B_s , (ii) the lengths of the gaps among these tasks, (iii) L , and (iv) the length of the second task of the last job in B_s , i.e. $H_s = \sum_{i=j_s}^{j_{s+1}-1} a_i + \sum_{i=j_s}^{j_{s+1}-2} G_i + L + b_{j_{s+1}-1}$, where $G_i \geq 0$ is the length of the gap between a_i and a_{i+1} . From Lemma 8, we have $G_i \leq |b_i - a_{i+1}| \leq b_i$, and thus

$$H_s \leq \sum_{i=j_s}^{j_{s+1}-1} (a_i + b_i) + L. \quad (2.1)$$

Observe that the algorithm starts a new block every time it cannot schedule the next upcoming job in steps Step 3i and Step 3ii. Therefore, there can be two reasons why j_{s+1} cannot be scheduled in B_s :

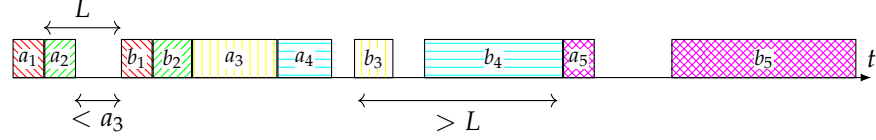


Figure 2.8: Reasons of starting a new block: a_3 does not fit between a_2 and b_1 , while the total completion times of the second tasks in the second block ($b_3 + b_4$) and the gap between them is larger than L .

(a) the gap between $a_{j_{s+1}-1}$ and b_{j_s} is smaller than $a_{j_{s+1}}$ or (b) the completion time of $b_{j_{s+1}-1}$ minus the starting time of b_{j_s} is larger than L , see Figure 2.8.

In case of (a), we have $\sum_{i=j_s+1}^{j_{s+1}} a_i + \sum_{i=j_s+1}^{j_{s+1}-1} b_i > L$, while in case of (b), we have $\sum_{i=j_s+1}^{j_{s+1}-1} a_i + \sum_{i=j_s}^{j_{s+1}-1} b_i > L$ (from Lemma 8). For each block, at least one of the previous inequalities holds, and neither of those inequalities contains any task occurring in any other inequality of another block. Hence, summing the valid inequalities for the first $s-1$ blocks, we have

$$(s-1)L < \sum_{i=1}^{j_s-1} (a_i + b_i) + a_{j_s}. \quad (2.2)$$

Applying Equation 2.1 and Equation 2.2 to bound the completion time C_j of a job j in B_s , we get

$$\begin{aligned} C_j &\leq \sum_{k=1}^{s-1} H_k + \sum_{i=j_s}^j (a_i + b_i) + L \leq \sum_{i=1}^j (a_i + b_i) + sL \\ &< 2 \sum_{i=1}^j (a_i + b_i) + L \leq 3C_j^{\text{OPT}_f}, \end{aligned}$$

where the last inequality follows from Lemma 2 and from $L < C_j^{\text{OPT}_f}$. Summing over all jobs, the Theorem follows. $\square$

Theorem 8. Algorithm 2 is a factor-1.5 approximation for $1|(p_j, L, p_j)| \sum C_j$.

Proof. Due to Lemma 7, it remains to prove that $C^{\text{ALG}} \leq 1.5C^{\text{OPT}}$.

Consider an arbitrary optimal solution. Note that, in the present case, the non-decreasing $a_j + b_j$ order is the same as the non-decreasing a_j order. Thus, we can use both lower bounds on the optimum described in Section 2.3.1. Also, $C_j^{\text{OPT}_f} = C_j^{\text{OPT}_s}$ follows directly from all delay times being fixed.

Consider jobs j and $j+1$ from the same block. From Lemma 8, and from $p_j \leq p_{j+1}$, we know that there is no gap between a_j and a_{j+1} in σ^{ALG} . Thus, the length of a block B_s can be expressed as

$$H_s = \sum_{k=j_s}^{j_{s+1}-1} p_k + L + p_{j_{s+1}-1}. \quad (2.3)$$

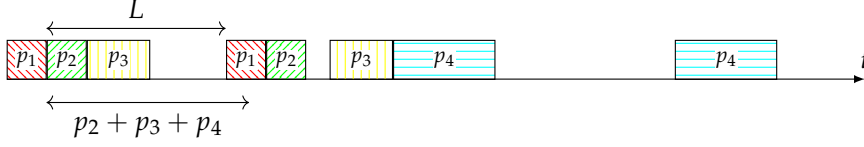


Figure 2.9: Since $p_2 + p_3 + p_4 > L$, job j_4 has to be scheduled in a new block.

Since j_s could not be scheduled in B_{s-1} , we have

$$\sum_{k=j_{s-1}+1}^{j_s} p_k > L, \quad s \geq 2, \quad (2.4)$$

see Figure 2.9.

In the remaining part of the proof we compare C_j and $C_j^{\text{OPT}_f}$, i.e. the completion time of job j in σ^{ALG} and the completion time of the j^{th} finishing (or starting) job in a fixed optimal schedule. For all $j > j_2$, we will prove $C_j \leq 1.5C_j^{\text{OPT}_f}$, but this inequality does not necessarily always hold true for $j = j_2$. However, with a more sophisticated analysis, we still manage to prove $C^{\text{ALG}} \leq 1.5C^{\text{OPT}}$ over the total of all jobs.

Let job j be a job in some block B_s , where $s \geq 2$ and $j \neq j_2$. Then the completion time C_j of j can be expressed as:

$$C_j = \sum_{i=1}^{s-1} H_i + \sum_{k=j_s}^j p_k + L + p_j.$$

Using Equation 2.3, we get

$$\begin{aligned} C_j &< \sum_{i=1}^{s-1} \left(\sum_{k=j_i}^{j_{i+1}-1} (p_k + L + p_{j_{i+1}-1}) \right) + \sum_{k=j_s}^j p_k + L + p_j \\ &\leq \sum_{i=1}^j p_i + \sum_{i=1}^{s-1} p_{j_{i+1}-1} + sL + p_j. \end{aligned}$$

Applying Inequality 2.4 once for blocks $B_1, \dots, B_{s-1}$, we get:

$$C_j < p_1 + 2 \sum_{k=2}^{j_s} p_k + \sum_{i=1}^{s-1} p_{j_{i+1}-1} + \sum_{k=j_s+1}^{j-1} p_k + 2p_j + L.$$

Applying it again for B_1 , and then using $p_{j_2-1} \leq p_j$ (since $j > j_2$), we have

$$\begin{aligned} C_j &< p_1 + 3 \sum_{k=2}^{j_2} p_k + 2 \sum_{k=j_2+1}^{j_s} p_k + \sum_{i=1}^{s-1} p_{j_{i+1}-1} + \sum_{k=j_s+1}^{j-1} p_k + 2p_j \\ &\leq p_1 + 3 \sum_{k=2}^{j_2} p_k + 2 \sum_{k=j_2+1}^{j_s} p_k + \sum_{i=2}^{s-1} p_{j_{i+1}-1} + \sum_{k=j_s+1}^{j-1} p_k + 3p_j \\ &\leq 3 \sum_{k=1}^j p_k. \end{aligned}$$

Using [Lemma 2](#) on this statement, we have $C_j < 1.5C_j^{OPT}$ and thus,

$$\sum_{j>j_2} C_j < 1.5 \sum_{j>j_2} C_j^{OPT_f}. \quad (2.5)$$

If j is in B_1 , we have $C_j = \sum_{k=1}^j p_k + L + p_j \leq C_j^{OPT_f}$, where the last inequality follows from [Lemma 3](#).

Now, if $j = j_2$, we then have $C_{j_2} = H_1 + 2p_{j_2} + L = \sum_{k=1}^{j_2} p_k + p_{j_2-1} + p_{j_2} + 2L$ (from [Equation 2.3](#)) and $C_{j_2}^{OPT_f} \geq \sum_{k=1}^{j_2} p_k + L + p_{j_2}$ (from [Lemma 3](#)). Thus, we have

$$\begin{aligned} \sum_{k=1}^{j_2} C_k &= (C_1 + C_{j_2}) + \sum_{k=2}^{j_2-1} C_k \\ &\leq \left(2p_1 + L + \sum_{k=1}^{j_2} p_k + p_{j_2-1} + p_{j_2} + 2L \right) + \sum_{k=2}^{j_2-1} C_k^{OPT_f} \\ &\leq 1.5 \left(C_1^{OPT_f} + C_{j_2}^{OPT_f} \right) + \sum_{k=2}^{j_2-1} C_k^{OPT_f} \leq 1.5 \sum_{k=1}^{j_2} C_k^{OPT_f} \end{aligned}$$

Therefore, $C^{ALG} = \sum_{j=1}^n C_j \leq 1.5 \sum_{j=1}^n C_j^{OPT_f} = 1.5C^{OPT}$, following from the previous statement and [Equation 2.5](#). $\square$

2.3.4 CTP with Related Processing and Delay Times

In this section we consider variants where at least one of the tasks has a processing time equal to the delay time. We first reuse [Algorithm 1](#):

Theorem 9. [Algorithm 1](#) is a factor-1.5 approximation for $1|(p_j, p_j, p_j)| \sum C_j$.

Proof. Due to [Lemma 5](#), it remains to prove that $C^{ALG} \leq 1.5C^{OPT}$. Observe that both first and the second tasks are in non-decreasing p_j order in σ^{ALG} . This means the completion time C_j of job j is at most $3 \sum_{i=1}^j p_i$. Thus, we have $C^{ALG} = \sum_{j=1}^n C_j \leq \sum_{j=1}^n \left(3 \sum_{i=1}^j p_i \right) \leq 1.5C^{OPT}$, where the last inequality follows from [Lemma 2](#). $\square$

Now, consider [Algorithm 3](#).

Theorem 10. [Algorithm 3](#) is a factor-2 approximation for $1|(a_j, p_j, p_j)| \sum C_j$.

Proof. It is straightforward that [Algorithm 3](#) runs in $\mathcal{O}(n \log n)$ time and always produces a feasible solution. We have $C^{ALG} \leq \sum_{j=1}^n \sum_{i=1}^j (a_j + 2p_j)$ from [Algorithm 3](#) and $C^{OPT} \geq \sum_{j=1}^n \sum_{i=1}^j (a_j + p_j)$ from [Lemma 2](#). Therefore, the Theorem follows. $\square$

If we modify the input, as well as the first step of [Algorithm 3](#), such that it takes instances of CTP with $L_j = a_j$, and sorts the jobs in non-decreasing $p_j + b_j$ order, we can approximate $1|(p_j, p_j, b_j)| \sum C_j$ with factor of 2. The proof is analogous to the proof of [Theorem 10](#).

Algorithm 3**Input:** a CTP instance with $L_j = b_j = p_j$ for each job j **Output:** a schedule σ for this instance

1. Sort the jobs in non-decreasing $a_j + p_j$ order. In the following, let the jobs be indexed in this order.
2. In this order, schedule the first task of each job right after the second task of the previously scheduled job has completed, and its second task according to the delay time.

Theorem 11. *Modified Algorithm 3 is a factor-2 approximation for $1|(p_j, p_j, b_j)| \sum C_j$.*

2.4 BI-OBJECTIVE APPROXIMATION

In this section, we give constant-factor (ρ_1, ρ_2) -approximations for all variants of the bi-objective $1|(a_j, L_j, b_j)|\{C_{\max}, \sum C_j\}$ problem for which we gave constant factor approximations of the $\sum C_j$ -objective in this chapter. Stein and Wein [118] defined two simple conditions on scheduling problems: *Truncation* (deleting jobs from a valid schedule results in a valid partial schedule) and *Composition* (a simple way of appending two valid partial schedules results in a valid schedule), proving the following:

Proposition 1 (Stein and Wein [118], Corollary 3). *For any scheduling problem satisfying the conditions Truncation and Composition, if there exists an α -approximation algorithm for the minimization of makespan and a β -approximation algorithm for the minimization of sum of completion times, there exists an $(\alpha(1 + \delta), \beta(\frac{\delta+1}{\delta}))$ -algorithm for any $\delta > 0$.*

Note that all considered coupled task problem variants fulfill these conditions. With this result in hand, we can now combine our β -approximation algorithms for the sum of completion times with previous α -approximation algorithms for the minimization of makespan to get (ρ_1, ρ_2) -approximations for all approximated $\sum C_j$ problems. We choose δ in such a way that the maximum of the two approximation factor $\max(\rho_1, \rho_2)$ is minimized. This is a common choice in bi-objective optimization as the goal is to get the best balanced result for both objectives simultaneously.

We give these results in Table 2.2. The first column of the table specifies the specific variant of $1|(a_j, L_j, b_j)|\{C_{\max}, \sum C_j\}$ to be approximated, with the variant identified by its job characteristics. The second column gives the (ρ_1, ρ_2) -approximation factor for each variant. As in our case ρ_1 always equals ρ_2 , we just give one value in this column. In the remaining columns we give the specific α and β values used in Proposition 1, with a reference to their origin, as well as our choice of

	$\rho_1 = \rho_2$	α	β	δ
(a, L_j, b)	6.5	3.5 ([8])	3 (Thm. 5)	6/7
(a_j, L, b_j)	6	3 ([7])	3 (Thm. 7)	1
$(a, L_j, b, b \leq a)$	5.5	3.5 ([8])	2 (Thm. 4)	4/7
(a_j, p_j, p_j)	5.5	3.5 ([8])	2 (Thm. 10)	4/7
(p_j, p_j, b_j)	5.5	3.5 ([8])	2 (Thm. 11)	4/7
(p_j, p_j, p_j)	4	2.5 ([8])	1.5 (Thm. 9)	3/5
$(1, L_j, 1)$	3.25	1.75 ([6])	1.5 (Thm. 6)	6/7
(p_j, L, p_j)	3	1.5 ([7])	1.5 (Thm. 8)	1

Table 2.2: Results on bi-objective (ρ_1, ρ_2) -approximation for CTP variants with $C_{\max}$ and $\sum C_j$ objectives

δ . The run time of this algorithm implied by Proposition 1 is the sum of the run times of both the α - and the β -approximation algorithms. As all used approximation algorithms for both $C_{\max}$ and $\sum C_j$ problems run in polynomial time, all (ρ_1, ρ_2) -approximations given in Table 2.2 can be computed in polynomial time as well.

2.5 CONCLUSION

In this chapter, we deal with the single machine coupled task scheduling problem, with the minimization of the total completion time as our objective function. Our work extends the complexity results of Chen and Zhang [29] and Kubiak [95] by introducing two new $\mathcal{NP}$ -hard variants, and provides several polynomial-time constant-factor approximation algorithms. To do this, we were able to modify several known algorithmic concepts used in coupled task makespan minimization, but some of our proofs on approximation factors required more sophisticated ideas. E.g., in the proof of Theorem 8, the original idea for the approximation factor only worked for jobs scheduled after a certain number of other jobs had already been processed, and a careful analysis of the approximation factor for the jobs before this cut-off was needed to get the result on hand.

We also give the first results on bi-objective approximation in the coupled task setting: We use a result from Stein and Wein [118], together with constant-factor approximations on the makespan objective taken from the literature, to give bi-objective constant-factor approximations for the problem of both minimizing the sum of completion time and the makespan simultaneously. We do this for all variants of CTP with the sum of completion times objective that we managed to approximate with a constant factor.

Although we did manage to provide approximation algorithms for several coupled task scheduling problem variants with the sum of

completion times objective, it is still unknown if there is a constant-factor approximation algorithm for a few important cases: for the most general case; for the cases where only one of the tasks has a fixed processing time; and for the (p_j, L_j, p_j) variant. This stems from the fact that all algorithms presented in this chapter make use of some unique ordering of the jobs implied by the job characteristics on either the task lengths or delay lengths. In the aforementioned cases, there exists no such unique ordering using only task lengths or delay lengths. Inapproximability results also are of interest for the problems presented in this chapter, as they do exist for most CTP variants with the makespan objective [7, 8]. To our best knowledge, there are no such results for CTP with the total completion time objective. We point to these two open questions as suggestions for future research.

PREEMPTIVE SCHEDULING WITH BATCH SETUP TIMES

3.1 INTRODUCTION

In this chapter, we investigate the problem of scheduling jobs with class dependent setup times on identical parallel machines while allowing preemption, with the objective of minimizing the makespan. This problem is denoted as $P|pmnt, setup = s_i|C_{\max}$ in classic Graham-Notation [57]. Formally, an instance of $P|pmnt, setup = s_i|C_{\max}$ consists of $m \in \mathbb{N}_{\geq 0}$ identical and parallel machines, a set J of $n \in \mathbb{N}_{\geq 0}$ jobs $j \in J$, a partition $\bigcup_{i=1}^c C_i = J$ of J into c nonempty and disjoint subsets $C_i \subseteq J$, called classes, a processing time $p_j \in \mathbb{N}_{\geq 0}$ for each jobs $j \in J$, and a setup time of $s_i \in \mathbb{N}_{\geq 0}$ for each class i . Allowing preemption means that each job j may be split into any number of job pieces $j_1, \dots, j_d$, with $p_{j_1}, \dots, p_{j_d} \in \mathbb{R}$ and $\sum_{k=1}^d p_{j_k} = p_j$, but no job pieces $j_k, j_{k'}$ of the same job j may be scheduled at the same time over all machines. We also say that a job j may not be *parallelized*. A job j is (fully) scheduled iff all of its job pieces are scheduled.

The objective of $P|pmnt, setup = s_i|C_{\max}$ is to find a feasible schedule σ which minimizes the makespan, i. e. the maximum completion time of any machine in σ . A schedule σ is feasible for an instance I of $P|pmnt, setup = s_i|C_{\max}$ iff all jobs are fully scheduled on at most m machines, on each machine $q \in [m]$, before any number of job pieces of the same class $i \in [c]$ are scheduled, there is a setup time s_i scheduled, and no job is parallelized. $P|pmnt, setup = s_i|C_{\max}$ is $\mathcal{NP}$ -hard already for $m = 2$ [105]. We thus focus on giving approximate solutions for this problem in polynomial time.

3.1.1 Related Work

The problem $P|pmnt, setup = s_i|C_{\max}$ was first introduced by Monma and Potts [105], where they not only proved the problems $\mathcal{NP}$ -hardness, but also gave a dynamic programming approach computing exact solutions to this problem in $\mathcal{O}(n^c)$. They were also the first to give a constant factor approximation algorithms for the problem [106], one with general approximation ratio of $5/3$ and run time in $\mathcal{O}(n + (m + c) \log(m + c)) \in \mathcal{O}(n \log(n))$, and another resembling the Wrap-Around-Rule introduced by McNaughton [104], with approximation ratio of $2 - 1/(\lfloor m/2 \rfloor + 1) = \lim_{m \rightarrow \infty} 2$ and run time in $\mathcal{O}(n)$. Deppert and Jansen [41] improved on this result, giving an

algorithm for $P|pmnt, setup = s_i|C_{\max}$ with approximation ratio of $3/2$ and slightly worse run time in $\mathcal{O}(n \log(n))$.

Notable, related variations of this problem include the problem $P|pmnt, setup = s_i|C_{\max}$ restricted to *single-job-batches*, i.e. $|C_i| = 1$ for all $i \in [c]$, the splittable version of the problem $P|split, setup = s_i|C_{\max}$, where jobs may not only be preempted, but also parallelized, and the non-preemptive version $P|setup = s_i|C_{\max}$. For the restriction of $P|pmnt, setup = s_i|C_{\max}$ to single-job-batches, Schuurman and Woeginger [113] gave a $(4/3 + \epsilon)$ -approximation, as well as a PTAS for the case of $s_i = s, \forall i \in [c]$. For $P|split, setup = s_i|C_{\max}$, Xing and Zhang [124] gave a $5/3$ -approximation algorithm. For $P|setup = s_i|C_{\max}$, there exists a series of constant-factor approximation algorithms [78, 101], as well as a PTAS [78]. Research on approximation algorithms for these three problem variants recently culminated when Jansen et al. [77] introduced EPTASs for each, based on a new n -fold integer program formulation and using an efficient algorithm for such integer programs [64].

For a concise history of approximating $P|pmnt, setup = s_i|C_{\max}$ and related problems, we refer the reader to the overview given by Deppert and Jansen [41]. Interestingly however, there does not yet exist a PTAS for general $P|pmnt, setup = s_i|C_{\max}$. Therefore, the $3/2$ -approximation algorithm by Deppert and Jansen [41] gives the currently best known approximation for this problem.

3.1.2 Our Contribution

We give a $(4/3 + \epsilon)$ -approximate algorithm for $P|pmnt, setup = s_i|C_{\max}$ with run time in $\mathcal{O}(n^2 \log(1/\epsilon))$. For any $\epsilon < 1/6$, this improves upon the previously best known approximation factor of $3/2$ of a polynomial time algorithm by Deppert and Jansen [41] for this problem.

ORGANIZATION OF THIS CHAPTER We first give some preliminary notions in Section 3.2, stating the basic notation used throughout this chapter, presenting how we structure instances of the problem, and introducing the important concept of “Batch Wrapping”. This is followed by Section 3.3, a rough technical overview of our core ideas. Our main results are then presented in multiple stages: We first give a $4/3T$ -dual approximation for “nice instances” in Section 3.4, followed by a structural analysis of general instances in Section 3.5. These ideas are then combined into the $(4/3 + \epsilon)$ -approximate algorithm for general instances of $P|pmnt, setup = s_i|C_{\max}$ in Section 3.6. We finish by giving a short conclusion and an outlook on future research in Section 3.7.

3.2 PRELIMINARIES

NOTATION We begin by giving some notation. A *job piece* j' of a job j is a new job j' with processing time $p_{j'} \leq p_j$ of the same class i as j . A job j is fully scheduled if all of its job pieces j' are scheduled, and the total processing times of its job pieces is p_j , i. e. $\sum_{j' \in J} p_{j'} = p_j$. Note that, by this definition, job j is a job piece of itself. We denote the total processing time of all jobs of a class as $P(C_i)$, i. e. $P(C_i) := \sum_{j \in C_i} p_j$. For readability purposes, we abuse the notation slightly and may use s_j to refer to the setup time s_i of class i with $p_j \in C_i$. We also may write s_x or C_x to refer to the setup time or job set of a class i_x , respectively. The load $L_\sigma(q)$ of a machine q is the sum of all setup times and all processing times of job pieces scheduled on q in σ . If σ is clear from context, we just write $L(q)$. The load $L(S)$ of a set of setup times and job pieces S is the total sum over all these elements, i. e. $L(S) := \sum_{s \in S} s$. We say a class i is “fully scheduled” on a machine q , w.r.t. some schedule σ , if all job (pieces) in C_i are scheduled on q in σ . By λ_i^σ , we denote the number of setup times s_i of class i scheduled in σ .

An instance $I' = (m', J, \dot{\bigcup}_{i=k}^c C_k, p'_1, \dots, p'_{|J|}, s_1, \dots, s_c)$ is a sub-instance of an instance $I = (m, J, \dot{\bigcup}_{i=k}^c C_k, p_1, \dots, p_{|J|}, s_1, \dots, s_c)$ iff $m' \leq m$ and $p'_j \leq p_j, \forall j \in J$. If I' is a sub-instance of a (sub-)instance I , it can be defined solely by $m' \leq m$ and $p'_1, \dots, p'_{|J|}$, as all other parameters are the same for both I and I' . Let T be a guess on the optimal makespan of an instance I of $P|pmnt, setup = s_i|C_{\max}$, and $\alpha_i := \lceil P(C_i)/(T - s_i) \rceil \geq \lfloor P(C_i)/(T - s_i) \rfloor =: \alpha'_i$.

Lemma 9. *For any schedule σ with makespan T , $\lambda_i^\sigma \geq \alpha_i, \forall i \in [c]$.*

Proof. In any schedule σ with makespan T , there is at most $(T - s_i)$ processing time of each class i scheduled on a single machine. Thus, there are at least $\lceil P(C_i)/(T - s_i) \rceil = \alpha_i$ different machines required to schedule the total processing time $P(C_i)$ of a class i in a feasible schedule. To be feasible, there must be at least one setup time s_i scheduled on each of the α_i machines, implying $\lambda_i^\sigma \geq \alpha_i, \forall i \in [c]$. $\square$

3.2.1 Structuring an Instance.

For some guess on the makespan T , we structure any instance of $P|pmnt, setup = s_i|C_{\max}$ by partitioning the classes of an instance into sets $\textcircled{1}, \dots, \textcircled{10} \subseteq [c]$ such that $[c] = \textcircled{1} \cup \textcircled{2} \cup \textcircled{3} \cup \textcircled{4} \cup \textcircled{5} \cup \textcircled{6} \cup \textcircled{7} \cup \textcircled{8} \cup \textcircled{9} \cup \textcircled{10}$, as defined in Table 3.1 and visualized in Figure 3.1, which we call “types”. For readability purposes, here, we assume that $T = 1$, and set $\text{exp} = [2/3, 1]$, $\text{med} = [1/3, 2/3]$, and $\text{chp} = [0, 1/3]$. This is a total partition of classes in any instance; each occurring class belongs to exactly one set from $\textcircled{1}$ to $\textcircled{10}$. This structuring will help us deal with

the different types of classes: For classes of the same type, we will use similar algorithmic ideas.

Set	s_i	$s_i + P(C_i)$	Set	s_i	$s_i + P(C_i)$
①	exp	$[1, m]$	⑥	med	$(1/3, 4/9]$
②	med	$[2 - s_i, m]$	⑦	chp	$[0, m]$
③	med	$(4/3, 2 - s_i)$	⑧	med	$(4/9, 1/2]$
④	med	$[1, 4/3]$	⑨	med $\cup$ exp	$(2/3, 5/6]$
⑤	med	$(1/2, 2/3]$	⑩	med $\cup$ exp	$(5/6, 1)$

Table 3.1: A total partition of the classes in an instance.

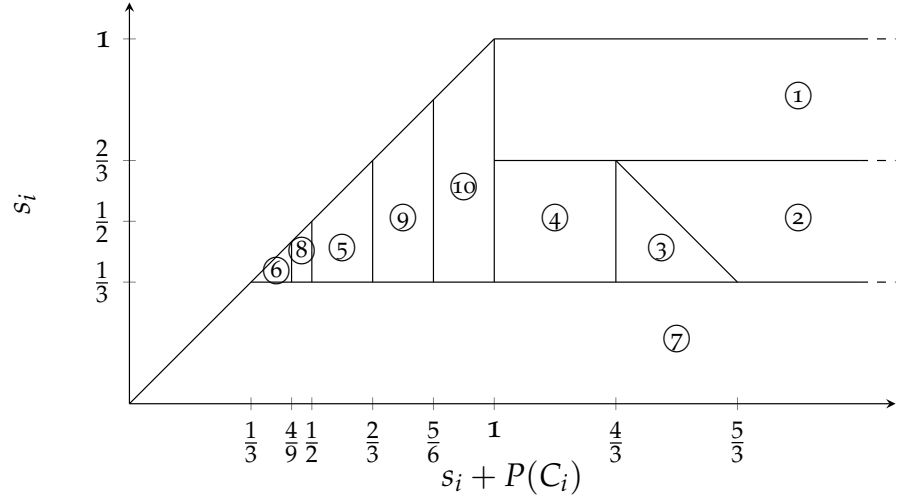


Figure 3.1: A graphical representation of the total partition of the classes of an instance.

3.2.2 Batch Wrapping

An algorithmic tool we will use repeatedly is called *Batch Wrapping*, i.e. the wrapping of *wrap sequences*, representing subsets of classes, into *wrap templates*, representing free space on machines, as defined by Deppert and Jansen [41] and based on ideas of McNaughton [104].

Definition 7 (Deppert and Jansen [41], Wrap Template). A *wrap template* is a list $\omega = (\omega_1, \dots, \omega_{|\omega|}) \in ([m] \times \mathbb{Q} \times \mathbb{Q})^*$ of triples $\omega_r = (u_r, a_r, b_r) \in [m] \times \mathbb{Q} \times \mathbb{Q}$ for $1 \leq r \leq |\omega|$ such that $0 \leq a_r < b_r$ and $u_r \leq u_{r+1}, \forall 1 \leq r \leq |\omega|$.

A wrap template represents some free space on machines $u_1, \dots, u_{|\omega|}$. Let $S(\omega) := \sum_{r=1}^{|\omega|} (b_r - a_r)$ denote the total period of time provided by wrap template.

Definition 8 (Deppert and Jansen [41], Wrap Sequence). A *wrap sequence* is a sequence

$$Q = (s_{i_1}, j_1^1, \dots, j_{n_1}^1, s_{i_2}, j_1^2, \dots, j_{n_2}^2, \dots, s_{i_k}, j_1^k, \dots, j_{n_k}^k)$$

where $C'_{i_\ell} = \{j_1^\ell, \dots, j_{n_\ell}^\ell\}$ with $n_\ell = |C'_{i_\ell}|$ is a set of job pieces of a class C_{i_ℓ} for some $i_\ell \in [c]$.

A wrap sequence now represents some subsets of classes in an instance. Deppert and Jansen [41] also describe an algorithm WRAP (Algorithm 4) that uses wrap templates to schedule wrap sequences. They prove the following properties:

Algorithm 4 WRAP (Q, ω)

Schedule all setup times and job pieces in the order given by Q into tuples ω_r in the order given by ω , beginning at a_r , ending and, if necessary, splitting job pieces at b_r . Setup times are not split, but instead not scheduled if they reach b_r . For every job piece to be scheduled at the beginning of a tuple ω_r , put a setup time of its class directly below a_r .

Property 1. Let Q be a wrap sequence containing a largest setup $s_{\max}^{(Q)}$ and ω be a wrap template with $L(Q) \leq S(\omega)$. Then, WRAP will place the last job (piece) of Q in a gap ω_r with $r \leq |\omega|$. If there was a free time of at least $s_{\max}^{(Q)}$ below each gap but the first, the load gets placed feasibly.

Property 2. If $L(Q) \leq S(\omega)$, then WRAP (Q, ω) has a run time of $\mathcal{O}(|Q| + |\omega|)$.

We also give a variant of WRAP that is more suited to scheduling larger jobs, which we call WRAPLJ (Algorithm 5). WRAPLJ has similar

Algorithm 5 WRAPLJ (Q, ω)

Schedule all setup times and job pieces in the order given by Q into tuples ω_r in the order given by ω , beginning at a_r , ending and, if necessary, splitting job pieces at b_r . Setup times are not split, but instead not scheduled if they reach b_r .

For every job piece to be scheduled at the beginning a_r of a triple ω_r , first schedule a setup time s_i of its class starting at a_r , and set the starting time of the job piece to $a_r + s_i$.

properties to WRAP, which we show in the following.

Property 3. Let Q be a wrap sequence, ω be a wrap template, and s_r be the first setup time placed on each machine $r \in |\omega|$ by WRAPLJ. If $L(Q) \leq S(\omega) - \sum_{r=2}^{|\omega|} s_r$ and $b_r - a_r > \max_{r \in |\omega|} s_r$, WRAPLJ will place the last job (piece) of Q in a gap ω_r with $r \leq |\omega|$.

Proof. As additional setup times not contained in Q are placed only at the start a_r of any ω_r for $r > 1$ in the wrap template, which is s_r by definition, the claim follows. $\square$

Property 4. If $L(Q) \leq S(\omega)$, then WRAPLJ (Q, ω) has a run time of $\mathcal{O}(|Q| + |\omega|)$.

To see the main difference between WRAPLJ and WRAP, consider a scenario where there is some interval of size S which we want to use for scheduling a wrap sequence Q . Using WRAP to schedule the jobs, the corresponding entry ω_r in a wrap template ω can have size at most $b_r - a_r \leq S - s_{\max}^{(Q)}$ to guarantee $s_{\max}^{(Q)}$ free space below a_r , allowing only jobs j with $p_j \leq S - s_{\max}^{(Q)}$ to be placed, with no regard for the actual setup time s_j of j . However, WRAP guarantees that all job pieces are placed above $s_{\max}^{(Q)}$, allowing us to place job pieces of jobs in Q below $s_{\max}^{(Q)}$ without worry of parallelization. In contrast, using WRAPLJ, for ω_r only $b_r - a_r \leq S$ must hold, allowing us to place any job j with $s_j + p_j \leq S$ in this interval. However, j can now start right after s_j , i. e. below $s_{\max}^{(Q)}$. We use both WRAP and WRAPLJ in the appropriate places to exploit their respective advantages. To easier establish feasibility of the schedules produced by WRAPLJ, we additionally show a property regarding parallelization of wrap templates and sequences when using this procedure.

Property 5. If $u_r < u_{r+1}, b_r = b_{r+1}, a_r = a_{r+1}, \forall r \in [1, |\omega| - 1]$ and $\max_{\ell \in [k], x \in n_\ell} s_{i_\ell} + j_x^\ell \leq b_r - a_r, \forall r \in [|\omega|]$, the schedule produced by WRAPLJ is never parallelized.

Proof. Let j_x^ℓ be a job piece that starts to be scheduled in $[a_r, b_r]$ by WRAPLJ. If j_x^ℓ is scheduled only in $[a_r, b_r]$, the claim follows. Otherwise, j_i^ℓ is interrupted at b_r , and then continued to be scheduled in $[a_{r+1}, b_{r+1}]$. As $u_r < u_{r+1}$, $[a_{r+1}, b_{r+1}]$ is on a different machine. Since $s_{i_\ell} + j_x^\ell \leq b_r - a_r = b_{r+1} - a_{r+1}$ by assumption, and s_{i_ℓ} is scheduled at a_{r+1} by WRAPLJ, the job piece of j_i^ℓ scheduled in $[a_{r+1}, b_{r+1}]$ is finished processing before the starting time of the job piece of j_i^ℓ in $[a_r, b_r]$, and the claim thus follows as well. $\square$

3.3 TECHNICAL OVERVIEW

For readability purposes, in this section, we informally describe our general procedure to obtain a $(4/3 + \varepsilon)$ -approximation algorithm for $P|\text{pmnt, setup} = s_i|C_{\max}$. We make use of a modified version of the classical dual approach by Hochbaum and Shmoys [69] (cf. [Section 1.1](#)), i. e. guess a $(1 + \varepsilon)$ -approximate makespan via binary search in time $\mathcal{O}(\log(1/\varepsilon))$, and our algorithm either returns a solution with makespan at most $4/3T$ for some makespan guess T , or allows us to conclude that $T < \text{OPT}$ if no feasible solution is returned.

For this, we begin with some structural properties: We begin by showing that for specific kinds of instances, called “nice instances” and consisting only of classes of types $\{\textcircled{1}, \dots, \textcircled{7}\}$ (and at most one class of $\textcircled{8}$), for any T , we can easily (in $\mathcal{O}(n)$) compute a $4/3T$ -approximate solution, or conclude that there exists no solution with makespan T . For later arguments, this algorithm places all jobs of $\textcircled{7}$ classes above $1/3T$. We then turn our attention to general instances, i.e. instances consisting of classes of all types $\{\textcircled{1}, \dots, \textcircled{10}\}$. Here, our first goal is to show that for any feasible solution with makespan T , there exists a solution with makespan at most $4/3T$ that has some specific properties. We do this by transforming any given solution with makespan T into one with makespan at most $4/3T$ with these properties. Specifically, we construct a feasible $4/3T$ solution by dividing the instance into a nice sub-instance, for which we already know how to compute a $4/3T$ -approximate solution, and “the rest”.

This rest consists only of classes of $\textcircled{8}$, $\textcircled{9}$, $\textcircled{10}$ and partial classes of $\textcircled{7}$. For this rest, we transform the given feasible solution with makespan T into one with makespan $4/3T$ where all classes of $\textcircled{9}$, $\textcircled{10}$, and pairs of classes of $\textcircled{8}$ are scheduled together on machines only with (partial) classes of $\textcircled{7}$. Specifically, jobs of these partial classes of $\textcircled{7}$ are either fully scheduled together with $\textcircled{9}$ or $\textcircled{10}$ classes in a feasible way, or placed in the interval $[0, 1/3T]$. This guarantees us that the parts of these $\textcircled{7}$ classes, which may contain job pieces also contained in the nice sub-instance, are never parallelized by the placement all jobs of $\textcircled{7}$ classes above $1/3T$ of our algorithm for nice (sub-)instances. Furthermore, the resulting $4/3$ -approximate solution is relatively easy to compute.

Finally, we construct a $(4/3 + \varepsilon)$ -approximate solution for any instance by doing the following: First, we guess the current makespan T . We then construct a solution with makespan at most $4/3T$ and our specified properties in time $\mathcal{O}(n^2)$, using the *minimum number of setup times* over all solutions with makespan at most $4/3T$ and the properties as specified above, if a feasible solution exists with makespan T . Thus, if there exists a solution with makespan T , we will find a $4/3T$ -approximate solution. If we are not able to compute a feasible $4/3T$ -approximate solution, this allows us to conclude that no solution with makespan T exists. The number of guesses needed to obtain a guess T with $\text{OPT} \leq T \leq (1 + \varepsilon) \text{OPT}$ is $\mathcal{O}(\log(1/\varepsilon))$, leading to the total run time of $\mathcal{O}(n^2 \log(1/\varepsilon))$.

3.4 A $4/3$ -APPROXIMATION FOR NICE INSTANCES

In this section we give a $4/3T$ -dual approximation for “nice instances”, i.e. instances with some computationally easy to handle properties. We define nice instances as follows.

Definition 9 (Nice Instances). For a makespan T we call an instance I *nice* iff $|\textcircled{8}| \leq 1$ and $|\textcircled{9}| = |\textcircled{10}| = 0$.

A nice instance I for a makespan T has an obligatory load of at least

$$\begin{aligned} L_{\text{nice}}(I, T) &:= \sum_{i \in \textcircled{1} \cup \textcircled{2}} \alpha'_i s_i + \sum_{i \in \textcircled{3}} 2s_i + \sum_{i \in \textcircled{4} \cup \textcircled{5} \cup \textcircled{6} \cup \textcircled{7} \cup \textcircled{8}} s_i + P(J) \\ &\leq \sum_{i \in [c]} (\alpha_i s_i + P(C_i)) \end{aligned}$$

where $P(J)$ is the total processing time of all jobs in the instance, and the last inequality follows from the definition of α_i . For a nice instance I and a makespan T , we also define $m_{\text{nice}}(I, T)$, which we show to be a lower bound on the minimum machine number necessary to schedule all classes. Let $R_3 \subseteq \textcircled{3}$, $R_5 \subseteq \textcircled{5}$, and $R_6 \subseteq \textcircled{6}$ such that $2\lfloor |\textcircled{3}|/2 \rfloor + |R_3| = |\textcircled{3}|$, $2\lfloor |\textcircled{5}|/2 \rfloor + |R_5| = |\textcircled{5}|$, and $3\lfloor |\textcircled{6}|/3 \rfloor + |R_6| = |\textcircled{6}|$ for an instance I and a makespan T . Moreover, let

$$\begin{aligned} m_{\text{nice}}(I, T) &= \sum_{i \in \textcircled{1} \cup \textcircled{2}} \alpha'_i + 3\left\lfloor \frac{|\textcircled{3}|}{2} \right\rfloor + |\textcircled{4}| + \left\lfloor \frac{|\textcircled{5}|}{2} \right\rfloor + \left\lfloor \frac{|\textcircled{6}|}{3} \right\rfloor + \\ &\quad \left\lceil \frac{r_{\text{nice}}(I, T)}{T} \right\rceil \end{aligned}$$

where

$$r_{\text{nice}}(I, T) = \sum_{i \in R_3} (2s_i + P(C_i)) + \sum_{i \in R_5 \cup R_6 \cup \textcircled{7} \cup \textcircled{8}} (s_i + P(C_i)).$$

If either I and/or T are clear from context, we may omit them and i. e. simply write L_{nice} or m_{nice} . We will use both the machine number m_{nice} as well as the load size L_{nice} to argue for the approximate schedulability of a nice instance. The following lemma establishes a relationship between these two values.

Lemma 10. *Let I be a nice instance and T be a makespan guess. If $L_{\text{nice}} \leq mT$, then $m_{\text{nice}} \leq m$.*

Proof. Here we prove the stronger statement that $\lceil \frac{1}{T} L_{\text{nice}} \rceil \geq m_{\text{nice}}$. If $L_{\text{nice}} \leq mT$, this then gives us $m = \lceil \frac{1}{T} mT \rceil \geq \lceil \frac{1}{T} L_{\text{nice}} \rceil \geq m_{\text{nice}}$. We may write

$$\begin{aligned} L_{\text{nice}} &= \sum_{i \in \textcircled{1} \cup \textcircled{2}} \left(\left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor s_i + P(C_i) \right) + \sum_{i \in \textcircled{3}} (2s_i + P(C_i)) + \\ &\quad \sum_{i \in \textcircled{4} \cup \textcircled{5} \cup \textcircled{6} \cup \textcircled{7} \cup \textcircled{8}} (s_i + P(C_i)) \end{aligned}$$

and

$$m_{\text{nice}} = \left\lceil \sum_{i \in \textcircled{1} \cup \textcircled{2}} \left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor + 3\left\lfloor \frac{|\textcircled{3}|}{2} \right\rfloor + |\textcircled{4}| + \left\lfloor \frac{|\textcircled{5}|}{2} \right\rfloor + \left\lfloor \frac{|\textcircled{6}|}{3} \right\rfloor + \frac{r_{\text{nice}}}{T} \right\rceil.$$

Thus, to show $\lceil \frac{1}{T} L_{\text{nice}} \rceil \geq m_{\text{nice}}$ it suffices to prove that

$$\frac{1}{T} L_{\text{nice}} \geq \sum_{i \in \textcircled{1} \cup \textcircled{2}} \left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor + 3 \left\lfloor \frac{|\textcircled{3}|}{2} \right\rfloor + |\textcircled{4}| + \left\lfloor \frac{|\textcircled{5}|}{2} \right\rfloor + \left\lfloor \frac{|\textcircled{6}|}{3} \right\rfloor + \frac{r_{\text{nice}}}{T}. \quad (3.1)$$

For all $i \in \textcircled{1} \cup \textcircled{2}$ we find

$$\begin{aligned} \frac{1}{T} \left(\left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor s_i + P(C_i) \right) &= \frac{1}{T} \left(\left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor s_i + \frac{P(C_i)}{T - s_i} (T - s_i) \right) \\ &\geq \frac{1}{T} \left(\left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor s_i + \left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor (T - s_i) \right) \\ &= \left\lfloor \frac{P(C_i)}{T - s_i} \right\rfloor \end{aligned}$$

and

$$\begin{aligned} \frac{1}{T} \sum_{i \in \textcircled{3} \setminus R_3} \underbrace{\left(s_i + \underbrace{s_i + P(C_i)}_{> \frac{4}{3}T} \right)}_{> \frac{1}{3}T} &\geq \frac{5}{3} |\textcircled{3} \setminus R_3| = \frac{5}{3} (|\textcircled{3}| - |R_3|) \\ &= \frac{5}{3} \cdot 2 \left\lfloor \frac{|\textcircled{3}|}{2} \right\rfloor \geq 3 \left\lfloor \frac{|\textcircled{3}|}{2} \right\rfloor \end{aligned}$$

and

$$\frac{1}{T} \sum_{i \in \textcircled{4}} \underbrace{(s_i + P(C_i))}_{> T} \geq |\textcircled{4}|$$

and

$$\begin{aligned} \frac{1}{T} \sum_{i \in \textcircled{5} \setminus R_5} \underbrace{(s_i + P(C_i))}_{> \frac{1}{2}T} &\geq \frac{1}{2} |\textcircled{5} \setminus R_5| = \frac{1}{2} (|\textcircled{5}| - |R_5|) \\ &= \frac{1}{2} \cdot 2 \left\lfloor \frac{|\textcircled{5}|}{2} \right\rfloor = \left\lfloor \frac{|\textcircled{5}|}{2} \right\rfloor \end{aligned}$$

and

$$\begin{aligned} \frac{1}{T} \sum_{i \in \textcircled{6} \setminus R_6} \underbrace{(s_i + P(C_i))}_{> \frac{1}{3}T} &\geq \frac{1}{3} |\textcircled{6} \setminus R_6| = \frac{1}{3} (|\textcircled{6}| - |R_6|) \\ &= \frac{1}{3} \cdot 3 \left\lfloor \frac{|\textcircled{6}|}{3} \right\rfloor = \left\lfloor \frac{|\textcircled{6}|}{3} \right\rfloor. \end{aligned}$$

By comparing the sides by type, this shows [Equation 3.1](#), which proves the claim. $\square$

We are now ready to describe the $4/3$ -approximation algorithm for nice instances I . Note that the algorithm places all jobs of classes in $\textcircled{7}$ only above $1/3T$, which will be important for later reasoning.

Theorem 12. *Let I be a nice instance with optimal objective value OPT , and T be a makespan guess. Then the following properties hold.*

- (i) If $m < m_{\text{nice}}$, then it holds that $T < \text{OPT}$.
- (ii) Otherwise a feasible schedule with makespan at most $\frac{4}{3}T$ can be computed in time $\mathcal{O}(n)$.

Proof. (i). We show that $T \geq \text{OPT}$ implies $mT \geq L_{\text{nice}}$, which in turn implies $m \geq m_{\text{nice}}$ (Lemma 10). Let $T \geq \text{OPT}$. Then there exists a feasible schedule σ with makespan T . Let $L(\sigma) = \sum_{i=1}^c (\lambda_i^\sigma s_i + P(C_i))$. Then

$$mT \geq L(\sigma) \geq \sum_{i=1}^c \alpha_i s_i + P(J) \geq L_{\text{nice}} ,$$

where the second inequality follows from Lemma 9.

(iii). Here, $m \geq m_{\text{nice}}$. Consider the following procedure:

1. Schedule all $i \in \textcircled{1} \cup \textcircled{2}$ on α'_i machines each, by scheduling s_i setup times starting at 0 on α'_i machines, and the jobs in arbitrary order one after another on the α'_i machines between s_i and $1 + (P(C_i) \bmod (T - s_i)) / \alpha'_i$, for each i , preempting each job at $1 + (P(C_i) \bmod (T - s_i)) / \alpha'_i$ and restarting it at s_i on the next machine if necessary.
2. Schedule pairs $i, i' \in \textcircled{3}$ together on 3 machines: Schedule s_i and $P(C_i) - (2/3 - s_i)$ processing time of i on the first machine, s_i and $2/3 - s_i$ processing time of C_i , as well as $s_{i'}$ and $2/3 - s_{i'}$ processing time of i' on the second machine, and $s_{i'}$ and $P(C_{i'}) - (2/3 - s_{i'})$ processing time of i' on the third machine.
3. Schedule each $i \in \textcircled{4}$ on a single machine.
4. Schedule all classes in $\textcircled{5}$ as pairs i, i' together on a single machine.
5. Schedule all classes in $\textcircled{6}$ as triples i, i', i'' together on a single machine.

After these steps, there is at most one class $i_3 \in \textcircled{3}$, one class $i_5 \in \textcircled{5}$, two classes $i_6, i'_6 \in \textcircled{6}$, at most one class $i_8 \in \textcircled{8}$, and all classes in $\textcircled{7}$ unscheduled in a nice instance. We first deal with all but the $\textcircled{7}$ classes. If i_3 exists, always schedule s_3 and $P(C_3) - (2/3 - s_3)$ processing time of i_3 on a single machine; this leaves exactly $P'(C_3) = 2/3 - s_3$ processing time of i_3 unscheduled. Let $P'(C_i) = P(C_i)$ for all $i \in \{i_5, i_6, i'_6, i_8\}$. Then, $s_i + P'(C_i) + s_{i'} + P'(C_{i'}) \leq 4/3$, for all $i, i' \in \{i_3, i_5, i_6, i'_6, i_8\}$. Thus, for any combination of 0 to 4 classes of $\{i_3, i_5, i_6, i'_6, i_8\}$, we can schedule the remaining processing time of these classes as given by P' on at most 2 machines. If all 5 classes in $\{i_3, i_5, i_6, i'_6, i_8\}$ are yet unscheduled, we schedule $1 < s_3 + P'(C_3) + s_6 + P'(C_6) \leq 4/3$ on one machine; the remaining processing time of the remaining classes $\{i_5, i'_6, i_8\}$ can then be scheduled again on at most 2 machines as before.

We do exactly this, scheduled such that one machine q_s starts processing two classes of $\{i_5, i_6, i'_6, i_8\}$ at 0, finishing at some time $a \leq 4/3$,

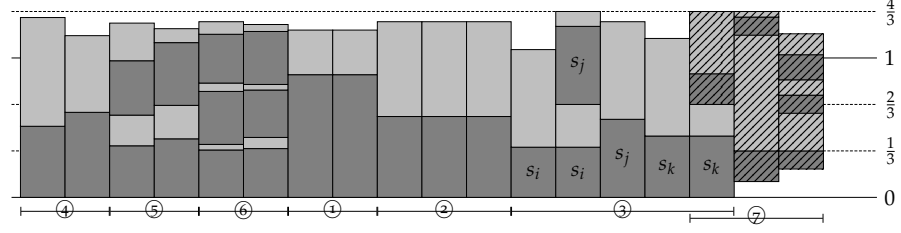
and the other machine q_e finishes processing two classes of $\{i_5, i_6, i'_6, i_8\}$ at $4/3$, starting at some time $b \geq 0$. Let M_{rem} be the set of all machines with nothing yet scheduled on them. If $L(q_s) + L(q_e) \leq 4/3$, we instead schedule all classes of $\{i_5, i_6, i'_6, i_8\}$ together on q_s starting at 0 and finishing at $a \leq 4/3$, leaving q_e empty and setting $b = 4/3$. To now schedule all classes $i \in \textcircled{7}$, we create a Wrap Template $\omega = (\omega_1, \dots, \omega_{|M_{\text{rem}}|+2})$, with $\omega_1 = (q_s, \max(a, 1/3T), 4/3T)$, $\omega_{|M_{\text{rem}}|+2} = (q_e, 1/3T, b)$, and $\omega_r = (q_{r-1}, 1/3, 4/3)$ for all $r = 2, \dots, |M_{\text{rem}}| + 1$, with q_r being the r -th machine in M_{rem} , as well as a Wrap Sequence Q with an entry $s_i, j_1^i, \dots, j_{n_i}^i$ for each $i \in \textcircled{7}$, $C_i = \{j_1^i, \dots, j_{n_i}^i\}$ in arbitrary order. Finally, we execute WRAP (Q, ω) .

There are enough machines to execute all steps without fail: As we use at most α_i setup times while scheduling each class $i \in \textcircled{1} \cup \dots \cup \textcircled{6} \cup \textcircled{8}$ to schedule these classes on some m' machines, distributing $> m'T$ load in the process, if at some point, there was no machine to schedule i , $m < m_{\text{nice}}$ by definition of m_{nice} , a contradiction. All m' machines finish for before $4/3T$: For $i \in \textcircled{1}$, $1 + (P(C_i) \bmod (T - s_i)) / \alpha'_i \leq 4/3$, as $s_i \geq 2/3T$, and for $i \in \textcircled{2}$, $1 + (P(C_i) \bmod (T - s_i)) / \alpha'_i \leq 4/3$ as well, as $s_i \geq 1/3T$ and $\alpha'_i \geq 2$. For $i \in \textcircled{3} \cup \dots \cup \textcircled{6} \cup \textcircled{8}$, each machine finishes before $4/3T$ by definition of the procedure. None of the jobs scheduled in this way are parallelized, as all jobs are either never preempted (for classes in $\textcircled{4}$, $\textcircled{5}$, $\textcircled{6}$ and $\textcircled{8}$), or preempted only at some time $> T$, and then continued at s_i until they finish (for classes in $\textcircled{1}$, $\textcircled{2}$ and $\textcircled{3}$), making parallelization impossible as $p_j \leq T - s_i$ for all jobs j in the instance. All these steps can be computed in total time $\mathcal{O}(n)$.

All classes in $\textcircled{7}$ are included in Q and scheduled via WRAP (Q, ω) . As

$$\begin{aligned} S(\omega) &\geq mT - \sum_{i \in \{\textcircled{1}, \dots, \textcircled{6}, \textcircled{8}\}} s_i + P(C_i) \\ &\geq m_{\text{nice}}T - \sum_{i \in \{\textcircled{1}, \dots, \textcircled{6}, \textcircled{8}\}} s_i + P(C_i) \\ &\geq \sum_{i \in \textcircled{7}} s_i + P(C_i) = L(Q) , \end{aligned}$$

Property 2 is fulfilled. As $s_{\max}^{(Q)} \leq 1/3$, and there is free time of $1/3$ for each ω_r with $r > 1$ by definition of ω , **Property 1** is fulfilled as well. The minimal continuous, non-overlapping free space in ω is $\min(S(\omega), 1)$; the largest processing time p_j of any job $j \in Q$ is at most $\min(L(Q), 1 - s_j)$; therefore, the partial schedule created by WRAP (Q, ω) is never parallelized. We refer to [Figure 3.2](#) for a visualization. $\square$

Figure 3.2: A $4/3$ -approximate solution of a nice instance.

3.5 STRUCTURAL INSIGHTS ON GENERAL INSTANCES

In the previous section, we have seen how to efficiently compute $4/3$ -approximate solutions for so-called “nice instances”, i.e. instances with arbitrary many classes only in types $\textcircled{1} \cup \dots \cup \textcircled{7}$, and at most one class in $\textcircled{8}$. We turn our attention now to general instances. In this section, we begin by showing some structural properties of such instances. The goal is to prove that, for every instance, if there exists a feasible solution for some makespan guess T , there also exists a $4/3T$ -approximate solution with some nice properties. Specifically, we aim to show that there every instance can be split into a nice instance and “the rest”. For the nice instance, the $4/3T$ -approximate solution is easy to compute; the rest will be shown to have some easy to compute properties as well. We do this by taking any optimal solution and successively transforming it into a $4/3T$ -approximate solution with the desired properties. In the algorithmic section, the hard part then becomes to compute such a solution without prior knowledge of the optimal, T -feasible solution.

For an instance I of $P|pmnt, \text{setup} = s_i|C_{\max}$, let σ^{OPT} be an optimal solution with makespan T . For readability purposes, we normalize the instance by dividing all relevant input parameters by T . At various points, we will combine even multiples of classes of $\textcircled{8}$ on even multiples of machines containing such classes. If, at such a step, the number of $\textcircled{8}$ classes to be combined is odd, we add the remaining $\textcircled{8}$ class, all setup times and job pieces scheduled on the same machine q as this class, as well as the machine q itself to a sub-instance $I_{8\text{rem}}$, which we will deal with afterwards.

In σ^{OPT} , there is a subset of machines with only classes of types $\textcircled{1} \cup \dots \cup \textcircled{7}$ scheduled on them. We call this set of machines M_{nice} , and the subset of setup times and job pieces scheduled on these machines S_{nice} . Let I_{nice} be the sub-instance induced by M_{nice} and all setup times and job pieces of jobs in S_{nice} . By definition of σ^{OPT} , it holds for I_{nice} that $L(S_{\text{nice}}) \leq |M_{\text{nice}}|$. Note that I_{nice} is not necessarily a nice instance yet, as even though it only consists of setup times and job pieces of nice classes from the original instance, not necessarily all job pieces of each nice class are included in I_{nice} . This may change the types of classes in the sub-instance. Specifically, each class $i \in \textcircled{1} \cup \dots \cup \textcircled{6}$ as

of yet partially included in I_{nice} , we aim to include in I_{nice} completely: This preserves the type of i for fixed makespan σ^{OPT} . For classes $i \in \textcircled{7}$, we may include only some job pieces of i in I_{nice} without violating the definition of a nice instance: Each subset of a class $i \in \textcircled{7}$ is itself in $\textcircled{7}$.

3.5.1 Non-Nice Classes with Multiple Setup Times

Now we begin to show how to also deal with machines containing non-nice types of classes. We start with those that use multiple setup times in σ^{OPT} . Let $M_{\lambda \geq 2}$ be the subset of machines in σ^{OPT} with classes $i \in \textcircled{8} \cup \textcircled{9} \cup \textcircled{10}$ scheduled on them and $\lambda_i^{\sigma^{\text{OPT}}} \geq 2$, i.e. all machines with classes of types $\textcircled{8}$, $\textcircled{9}$ or $\textcircled{10}$ that use more than one setup in σ^{OPT} . We first remove all setup times and job pieces from $M_{\lambda \geq 2}$, and then schedule all such classes of $\textcircled{8}$ as pairs on one machine, and each such class of $\textcircled{9}$ and $\textcircled{10}$ alone on one machine of $M_{\lambda \geq 2}$, placing all these classes of $\textcircled{8}$, $\textcircled{9}$ and $\textcircled{10}$. Let $M'_{\lambda \geq 2} \subseteq M_{\lambda \geq 2}$ be the set of machines used in the process of rescheduling above. Then the total sum L_{rest} of all setup times and job pieces S_{rest} scheduled on $M_{\lambda \geq 2}$ in σ^{OPT} , but not scheduled on $M'_{\lambda \geq 2}$ is at most $L(M_{\lambda \geq 2}) - L(M'_{\lambda \geq 2}) - 1/3|M'_{\lambda \geq 2}|$, as for each class $i \in \textcircled{8} \cup \textcircled{9} \cup \textcircled{10}$ scheduled on $M'_{\lambda \geq 2}$, there is now exactly one setup time scheduled on a machine in $M'_{\lambda \geq 2}$, but there were at least $\lambda_i^{\sigma^{\text{OPT}}} \geq 2$ setup times scheduled on $M_{\lambda \geq 2}$ scheduled in σ^{OPT} ; by definition of the classes i , each such setup time is $> 1/3$.

By definition of σ^{OPT} , $L(M_{\lambda \geq 2}) \leq |M_{\lambda \geq 2}|$. As it now holds for the load of each machine $q \in M'_{\lambda \geq 2}$ that $L(q) \geq 2/3$ by definition of the types scheduled on them, we can conclude that $L_{\text{rest}} = L(M_{\lambda \geq 2}) - L(M'_{\lambda \geq 2}) - 1/3|M'_{\lambda \geq 2}| \leq |M_{\lambda \geq 2}| - 2/3|M'_{\lambda \geq 2}| - 1/3|M'_{\lambda \geq 2}| = |M_{\lambda \geq 2}| - |M'_{\lambda \geq 2}|$. All setup times and job pieces in L_{rest} are from classes $i \in \textcircled{1} \cup \dots \cup \textcircled{7}$ by definition. We set $S_{\text{nice}} = S_{\text{nice}} \cup S_{\text{rest}}$ and $M_{\text{nice}} = M_{\text{nice}} \cup (M_{\lambda \geq 2} \setminus M'_{\lambda \geq 2})$, preserving $L(S_{\text{nice}}) \leq |M_{\text{nice}}|$.

3.5.2 Constructing a Nice Sub-Instance

After dealing with all non-nice types of classes that use multiple setup times in σ^{OPT} , we are only now left to deal with the non-nice types that use exactly one setup time in σ^{OPT} . Let $M_{\lambda=1}$ be the set of machines with classes $i \in \textcircled{8} \cup \textcircled{9} \cup \textcircled{10}$ and $\lambda_i^{\sigma^{\text{OPT}}} = 1$.

Note that $M = M_{\text{nice}} \dot{\cup} M_{\lambda=1} \dot{\cup} M'_{\lambda \geq 2}$, and that for all $q \in M_{\lambda=1}$ that have a class $i \in \textcircled{9} \cup \textcircled{10}$ scheduled on them in σ^{OPT} , all remaining setup times and job pieces in σ^{OPT} belong to classes in $\textcircled{7}$: The setup times of all other types of classes are already too large to be scheduled together with i on q . Let $M_9 \subseteq M_{\lambda=1}$ be the set of machines with a class of $\textcircled{9}$, $M_{10} \subseteq M_{\lambda=1}$ the set of machines with a class of $\textcircled{10}$ scheduled on them in σ^{OPT} . For all machines $q \in M_{\lambda=1}$ that have at least one class $i \in \textcircled{8}$ scheduled on them in σ^{OPT} , there is either another class $i' \in \textcircled{8}$ scheduled on q , or not. Let $M_{\text{dual}} \subseteq M_{\lambda=1}$ be

the set of machines for which the former, $M_{\text{single}} \subseteq M_{\lambda=1}$ the set of machines for which the latter is true. All remaining setup times and job pieces in σ^{OPT} scheduled on machines in M_{dual} belong to classes in ⑦ by the same argument as above. All remaining setup times and job pieces in σ^{OPT} scheduled on machines in M_{single} belong to classes in ① $\cup \dots \cup$ ⑦ by definition of the machine set. Note that $M_{\lambda=1} = M_{\text{single}} \dot{\cup} M_{\text{dual}} \dot{\cup} M_9 \dot{\cup} M_{10}$.

We first want to finish making the sub-instance induced by S_{nice} and M_{nice} an actual nice instance. This may not be the case yet: For some classes $i \in \textcircled{1} \cup \dots \cup \textcircled{6}$, there may be some job pieces not yet included in S_{nice} , meaning that in the sub-instance, these i are not complete. The only job pieces of such classes not included in S_{nice} must be scheduled on machines $q \in M_{\text{single}}$, as argued above. We now add these job pieces, and all other classes that are scheduled over multiple machines of M_{single} in σ^{OPT} , to S_{nice} . Let $M'_{\text{single}} \subseteq M_{\text{single}}$ be the set of machines where at least one setup time and a job piece j of a class $i \in \textcircled{1} \cup \dots \cup \textcircled{6}$ is scheduled in σ^{OPT} , but not the whole class, i.e. $M'_{\text{single}} = \{q \in M_{\text{single}} \mid \exists i \in \textcircled{1} \cup \dots \cup \textcircled{6}, j \in C_i : \exists p_j, \sigma^{\text{OPT}}(j) = q, P(C_i) > p_j > 0\}$. Note that the rest of class i may be scheduled on machines $q' \in M'_{\text{single}}$ or included in S_{nice} .

Let i_8, i'_8 denote the two ⑧ classes scheduled on machines $q, q' \in M_{\text{single}}$, respectively. Now, we do the following modifications: While there exists a class $i \in \textcircled{1} \cup \dots \cup \textcircled{6}$ such that two job pieces $j, j' \in C_i$ are scheduled on two different machines $q, q' \in M'_{\text{single}}$, schedule both classes i_8, i'_8 together on q . Let S_{rest} be the set of all setup times and job pieces not belonging to class i , i_8 or i'_8 that have been previously scheduled on q and q' . Add all job pieces of class i , only one setup time s_i , and S_{rest} to S_{nice} , and q' to M_{nice} . The machine q is added to M_{dual} . The total load of the setup times and job pieces added to S_{nice} in every step is at most $7/9$, as the total load of all setup times in job pieces on machines q and q' was at most 2 in σ^{OPT} , and we did both not include classes i_8, i'_8 of load at least $4/9$ each, as well as discard one of the at least two setup times $s_i \geq 1/3$ that were previously scheduled on q and q' as well. As q' is added to M_{nice} , $L(S_{\text{nice}}) \leq |M_{\text{nice}}|$ is preserved.

Now, for each class $i \in \textcircled{1} \cup \dots \cup \textcircled{6}$ with more than one setup time s_i in σ^{OPT} , there is at most one machine $q \in M'_{\text{single}}$ with job pieces of i scheduled on it in σ^{OPT} . While there exist two different such classes i, i' with job pieces on machines $q, q' \in M'_{\text{single}}$, we again schedule both classes i_8, i'_8 together on q and add q to M_{dual} . We add all job pieces pieces of classes i and i' , but none of their setup times, and again S_{rest} to S_{nice} , as well as the machine q' to M_{nice} . The total load added to S_{nice} is at most $2 - 2(4/9) - 2(1/3) = 2/9$ by the same arguments as above, and by adding q' to M_{nice} , $L(S_{\text{nice}}) \leq |M_{\text{nice}}|$ is preserved. Afterwards, if $|M'_{\text{single}}| \neq 0$, $|M'_{\text{single}}| = 1$ by the pairing above, and we remove $q \in M'_{\text{single}}$ from this set, add the class i_8 of ⑧ on q to $I_{8\text{rem}}$,

and add all setup times and job pieces which scheduled on q in σ^{OPT} , except i_8 , to S_{nice} and q to M_{nice} .

Now, I_{nice} actually constitutes a nice instance: All classes of types $\textcircled{1} \cup \dots \cup \textcircled{6}$ that are included in S_{nice} are included fully. It is only left to argue that $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$, which implies that we can compute a $4/3$ -approximation for this sub-instance in time $\mathcal{O}(n)$ by way of [Lemma 10](#) and [Theorem 12, \(ii\)](#). Since we already know that $L(S_{\text{nice}}) \leq |M_{\text{nice}}|$, it is enough to show $L_{\text{nice}}(I_{\text{nice}}) \leq L(S_{\text{nice}})$. This is not necessarily the case, as the number of setup times in $L_{\text{nice}}(I_{\text{nice}})$ may be higher than in S_{nice} . We however show that this is in fact the case. Let λ'_i be the number of setup times of class i in S_{nice} , and J the set of all job pieces in S_{nice} . The expression

$$\begin{aligned} L_{\text{nice}}(I_{\text{nice}}) &= P(J) + \sum_{i \in \textcircled{1} \cup \textcircled{2} \cup \textcircled{4}} \alpha'_i s_i + \sum_{i \in \textcircled{3} \cup \textcircled{5} \cup \textcircled{6}} \alpha_i s_i + \sum_{i \in \textcircled{7}} s_i \\ &\leq P(J) + \sum_{i \in \textcircled{1} \cup \dots \cup \textcircled{7}} \lambda'_i = L(S_{\text{nice}}) \end{aligned}$$

is true if,

$$\alpha'_i \leq \lambda'_i, \forall i \in \textcircled{1} \cup \textcircled{2} \cup \textcircled{4}, \quad \alpha_i \leq \lambda'_i, \forall i \in \textcircled{3} \cup \textcircled{5} \cup \textcircled{6},$$

$$\text{and } 1 \leq \lambda'_i, \forall i \in \textcircled{7}.$$

The only time we do not add all $\lambda_i^{\sigma^{\text{OPT}}}$ setup times of a class i to S_{nice} is when dealing with partial nice classes on machines in M_{single} . For all $i \in \textcircled{1}$, there actually is no job part on any machine $q \in M'_{\text{single}}$ in σ^{OPT} , as the setup of any such class and a full class of $\textcircled{8}$ do not fit together on a single machine. Thus, all setup times of i in σ^{OPT} are added to S_{nice} , and $\lambda'_i = \lambda_i^{\sigma^{\text{OPT}}} \geq \alpha'_i$. For all i , let $P'(C_i)$ be the load scheduled on all machines except M'_{single} in σ^{OPT} , and $P''(C_i) = P(C_i) - P'(C_i)$ be the load on machines in M'_{single} in σ^{OPT} . Then, for all $i \in \textcircled{2}$,

$$\begin{aligned} \lambda'_i &\geq \lceil P'(C_i)/(1 - s_i) \rceil + \lfloor P''(C_i)/(10/9 - 2s_i) \rfloor \\ &> \lfloor P'(C_i)/(1 - s_i) \rfloor + \lfloor P''(C_i)/(1 - s_i) \rfloor = \alpha'_i, \end{aligned}$$

as there is at most $(10/9 - 2s_i)$ processing time of $P''(C_i)$ on two machines in M'_{single} , and there is a setup time s_i added to S_{nice} for every two such machines. The last inequality follows from the fact that $(1 - s_i) > (10/9 - 2s_i)$ for $s_i \geq 1/3$, which is the case for $i \in \textcircled{2}$. For $i \in \textcircled{3}$, $\alpha_i = 2$ by definition. If $P'(C_i) > (1 - s_i)$, $\lambda'_i \geq 2$ follows immediately for $i \in \textcircled{3}$. Otherwise, $P''(C_i) > 1/3$ holds by definition of $\textcircled{3}$, and thus is scheduled over at least 2 machines in M_{single} as at most $1 - 4/9 - 1/3 = 2/9$ processing time of i can be scheduled on a machine in M_{single} for any $s_i \geq 1/3$. In any case, $\lambda'_i \geq 2$. For all $i \in \textcircled{4}$, $\lambda'_i \geq \alpha'_i$, as $\alpha'_i = 1$ and $\lambda'_i \geq 1$ by the fact that we always add at least one setup time per class to S_{nice} . The same holds for all $i \in \textcircled{5} \cup \textcircled{6}$ and $\lambda'_i \geq \alpha_i$, as $\alpha_i = 1$ for such i , as well as for all $i \in \textcircled{7}$. Thus, I_{nice} is a nice instance, and we can calculate a $4/3$ -approximation in time $\mathcal{O}(n)$ for it.

3.5.3 Combining Classes of Type Eight and Nine

Now, for each remaining machine in the set $M_{\lambda=1}$, we not only know that there is either exactly one class $i \in \textcircled{8} \cup \textcircled{9} \cup \textcircled{10}$ or exactly two classes $i \in \textcircled{8}$ fully scheduled on it in σ^{OPT} , but also that for each machine with one $i \in \textcircled{9} \cup \textcircled{10}$ or two $i, i' \in \textcircled{8}$ on it, all other job pieces scheduled on this machine belong to some classes of $\textcircled{7}$; and for each machine with one $i \in \textcircled{8}$ on it, there is at most one class $i' \in \textcircled{5} \cup \textcircled{6}$ fully scheduled on it (as no other nice class fully fits on such a machine in σ^{OPT}), and all other job pieces scheduled on this machine belong to some classes of $\textcircled{7}$. We proceed by combining classes of $\textcircled{8}$ and $\textcircled{9}$ efficiently. While $M_{\text{single}} \neq \emptyset$ and $M_9 \neq \emptyset$, let $q \in M_{\text{single}}$, $q' \in M_9$ and i_8 the $\textcircled{8}$ -class on q , i_9 the $\textcircled{9}$ -class on q' , and S_{rest} the set of all other setup times and job pieces previously scheduled on q and q' . Schedule both i_8 and i_9 on q . The new load of q then is always at least $4/9 + 2/3 = 10/9 > 1$, and at most $1/2 + 5/6 = 4/3$. Thus, $L(S_{\text{rest}}) \leq 2 - 10/9 < 1$. As items in S_{rest} amount to either full classes of $\textcircled{5} \cup \textcircled{6}$ and/or are job pieces of some classes of $\textcircled{7}$, adding them to S_{nice} and q' to M_{nice} preserves both the nice property of I_{nice} , as well as $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$. If after this process, $M_9 = \emptyset$, we are done. Otherwise, $M_{\text{single}} = \emptyset$.

Now, while $M_{\text{dual}} \neq \emptyset$ and $|M_9| \geq 2$, let $q \in M_{\text{dual}}$, $q', q'' \in M_9$ and i_8, i'_8 the $\textcircled{8}$ -classes on q , i_9, i'_9 the $\textcircled{9}$ -classes on q', q'' , respectively. Let S_{rest} be the set of all other setup times and job pieces previously scheduled on q, q' and q'' . Similarly, schedule both i_8 and i_9 on q , i'_8 and i'_9 on q' . Then the new load of both q and q' is always at least $4/9 + 2/3 = 10/9 > 1$, and at most $1/2 + 5/6 = 4/3$. With the same argument as above, we add S_{rest} to S_{nice} , and q'' to M_{nice} , preserving the aforementioned properties.

If, after this, $M_9 \neq \emptyset$ and $M_{\text{dual}} \neq \emptyset$, then $|M_9| = 1$. For the remaining $\textcircled{9}$ -class i_9 on a machine $q \in M_9$, we schedule it together on q with a $\textcircled{8}$ -class i_8 taken from a machine $q' \in M_{\text{dual}}$. We schedule S_{rest} and the other $\textcircled{8}$ -class i'_8 on q' , and add q' to M_{single} . The resulting schedule might not be feasible anymore, but we will restore feasibility later while dealing with all machines in M_{single} . Note that now, $L(q') \leq 1 - 4/9 + 1/3 = 8/9$, as we have removed i_8 with load at least $4/9$ from q' , and added the total load of setup times and jobs removed from q to q' , which is at most $1/3$.

Now, either $M_9 = \emptyset$ or $M_{\text{single}} = M_{\text{dual}} = \emptyset$, i. e. we have either dealt with all $\textcircled{9}$ or all $\textcircled{8}$ classes. Since $\textcircled{10}$ is the only other non-nice type to be distributed, we make a case distinction: We show how to distribute classes of $\textcircled{10}$ together only with classes of $\textcircled{8}$, or together only with classes of $\textcircled{9}$, in a way that results in a modified schedule that is $4/3$ -approximation with some easy to compute properties. As either case will also be applicable if neither classes of $\textcircled{8}$ or $\textcircled{9}$ are remaining, this then shows how to distribute all non-nice types.

3.5.4 Classes of Type Eight Remain

We start by constructing a bipartite graph, with a vertex for each class of ⑧ on $M_{\text{single}} \cup M_{\text{dual}}$ and ⑩ in M_{10} , and an edge between two vertices iff one vertex represents a class of ⑧, the other a class of ⑩, and the combined load both classes is lesser or equal than $4/3$. We compute a maximal matching for this graph, iteratively scheduling all classes of $i_8 \in \text{⑧}$ and $i_{10} \in \text{⑩}$ corresponding to the vertices of an edge in the maximal matching together on the machines $q' \in M_{10}$ where i_{10} was scheduled previously, and remove q' from M_{10} . Let S_{rest} now be the set of all setup times and job pieces previously scheduled on q and q' , except for i_8 and i_{10} . If i_8 was previously scheduled on a machine $q \in M_{\text{single}}$, S_{rest} is added to S_{nice} , and q is removed from M_{single} , and added to M_{nice} . As all items in S_{rest} are either parts of ⑦ classes, or include a full ⑤ or ⑥ class, and their total load is at most $2 - (5/6 + 4/9) = 13/18 < 1$ by definition of i_8 and i_{10} , this preserves $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$.

If i_8 was instead previously scheduled on a machine $q \in M_{\text{dual}}$, we schedule all items in S_{rest} together with the other ⑧-class i'_8 on q , removing q from M_{dual} and adding it to M_{single} . The resulting schedule might not be feasible anymore, but we will restore feasibility later while dealing with all other machines in M_{single} . Again, by definition of i_8 and i_{10} , $L(q) \leq 2 - (5/6 + 4/9) = 13/18$. By definition of the graph and the minimum load of classes i_8 and i_{10} , respectively, machine q' now has load at least 1 and at most $4/3$. It then follows for all combinations of remaining classes of ⑧ and ⑩ after this process, that their combined load is more than $4/3$: Otherwise, the matching would not have been maximal.

Now consider the remaining machines with classes of ⑧ and ⑩ with their scheduling not modified by the process above. We start by adding all setup times and job pieces scheduled on machines $q \in M_{\text{dual}} \cup M_{10}$, the ⑧, ⑩ and (partial) ⑦ classes, to a set $S_{8,10}$, which will store all setup times in job pieces not included in the nice sub-instance, and set $M_{8,10} = M_{\text{dual}} \cup M_{10}$, which represents all machines not yet used at another time in the modified schedule. Later, we will show that we can use a combination of direct placement and Batch-Wrapping to place all these setup times and job pieces in $S_{8,10}$ on the machines of $M_{8,10}$ in a feasible way below $1/3$. The construction of $S_{8,10}$ and $M_{8,10}$ will be done in a way that makes this possible. We proceed by looking at M_{single} . We aim to combine two ⑧ classes on machines in M_{single} , with the addition of some job pieces of ⑦ classes, on one of their machines, such that the other machine and the remaining job pieces can be added to I_{nice} while preserving computability of the nice $4/3$ -approximation. Recall that, when adding (partial) ⑦ classes to I_{nice} , maintaining $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$ is enough to maintain the niceness of I_{nice} . Let S_{rest} be the set of setup times and job pieces on

q and q' not belonging to i_8 or i'_8 , i. e. some (partial) ⑦ classes or full ⑤ or ⑥ classes, for some $q, q' \in M_{\text{single}}$ with i_8 scheduled on q and i'_8 scheduled on q' in σ^{OPT} . We first consider some “easy” combinations.

Case 1: $\exists q, q' \in M_{\text{single}} : L(S_{\text{rest}}) \leq 1$.

Adding i_8, i'_8 to $S_{8,10}$, q to $M_{8,10}$, S_{rest} to S_{nice} and q' to M_{nice} is enough: This preserves $L_{\text{nice}}(S_{\text{nice}}) \leq |M_{\text{nice}}|$. We do this for all pairs of q, q' where this is applicable. Afterwards, either $|M_{\text{single}}| \leq 1$, or it holds for all pairs $q, q' \in M_{\text{single}} : L(S_{\text{rest}}) > 1$.

Case 2: $\exists q, q' \in M_{\text{single}}, i, i' \in \textcircled{5} \cup \textcircled{6} : \sigma^{\text{OPT}}(i) = q \wedge \sigma^{\text{OPT}}(i') = q'$.

We may also simply add i_8, i'_8 to $S_{8,10}$ and q to $M_{8,10}$. By definition of nice instances, the instance I' consisting of all job pieces in S_{rest} and q' is itself a nice instance, for which $m_{\text{nice}}(I') = 1$. We know that, since $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$, $m_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$ by [Lemma 10](#). Thus, $m_{\text{nice}}(I_{\text{nice}} \cup I') \leq |M_{\text{nice}}| + 1$ by definition of m_{nice} , and [Theorem 12, \(ii\)](#) holds for I_{nice} after adding S_{rest} to I_{nice} and q' to M_{nice} . We again do this for all pairs of i_8, i'_8 where this is applicable. This schedules at least all but one such machine $q \in M_{\text{single}}$. We add q and all setup times and job pieces scheduled on it in σ^{OPT} to $I_{8\text{rem}}$.

Case 3: $\exists q, q' \in M_{\text{single}}, i, i' \in \textcircled{7} : s_i + P(C_i) \geq 1/2, s_{i'} + P(C_{i'}) \geq 1/2$ and i, i' are fully scheduled on q, q' , respectively.

By definition of σ^{OPT} , we know that $s_i + P(C_i) \leq 5/9$ and $s_{i'} + P(C_{i'}) \leq 5/9$. We fully schedule i and i' on q' (with completion time at most $10/9$), and put $S_{\text{rest}} \setminus \{i, i'\}$ into $S_{8,10}$, q into $M_{8,10}$. Note that, by definition of i, i' , $L(S_{\text{rest}} \setminus \{i, i'\}) \leq 1/9$.

By doing this for all such pairs where this is applicable, there is at most one such $q \in M_{\text{single}}$ with a class i as defined above remaining afterwards. We add i_8, i and $S_{\text{rest}}(q)$ to $S_{8,10}$ and q to $M_{8,10}$. Note that $L(i) + L(i_8) \leq 1$.

After this, it hold for all pairs of machines $q, q' \in M_{\text{single}}$ that only setup times and job pieces of ⑦ are in S_{rest} , and $L(S_{\text{rest}}) > 1$. Furthermore, $L(i_8) + L(i_{10}) > 4/3$, for all $i_8 \in \textcircled{8} \cup S_{8,10}, i_{10} \in \textcircled{9} \cup S_{8,10}$: Otherwise, our matching of ⑧ and ⑩ classes would not have been maximal. Let i_{10} be the class with smallest load over all classes in $\textcircled{9} \cup S_{8,10}$, let $L(i_{10}) = 8/9 - Y'$ for some $Y' \in \mathbb{Z}$, and let $Y = \max(0, Y')$. This definition implies that $0 \leq Y \leq 1/18$. Then it holds for all $i_8 \in \textcircled{8} \cap S_{8,10}$ that $L(i_8) \geq 4/9 + Y$: Otherwise, $L(i_{10}) + L(i_{10}) \leq 4/3$. Note that $L(i_8) + L(i'_8) \geq 8/9 + 2Y$ for all such classes i_8, i'_8 , and thus, $L(S_{\text{rest}}) \leq 10/9 - 2Y$.

Let $S_{\text{rest}}(q), S_{\text{rest}}(q') \subseteq S_{\text{rest}}$ be the subsets of S_{rest} originally scheduled on q and q' , respectively. Let $L(S_{\text{rest}}(q)) = 1/2 + x$, $L(S_{\text{rest}}(q')) = 1/2 + x'$, where $-1/18 \leq x, x' \leq 1/18 - Y$ by definition of q, q' . By previous case distinction, we know that $x + x' > 0$ for all such pairs. In the

following, we will always add q' to M_{nice} , and add q to $M_{8,10}$ or directly schedule some classes on it. To preserve $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$, it is then enough to add at most $(L(S_{\text{rest}}(q)) - \max(0, x)) + (L(S_{\text{rest}}(q')) - \max(0, x'))$ load to S_{nice} . We show how to do this by a full case distinction on the types of classes and jobs in $S_{\text{rest}}(q)$ and $S_{\text{rest}}(q')$. If $|M_{\text{single}}|$ is not even, we remove a machine q from M_{single} , and add it, as well as all setup times and job pieces scheduled on it in σ^{OPT} , to $I_{8\text{rem}}$.

Now, while $|M_{\text{single}}| \neq \emptyset$, let $q, q' \in M_{\text{single}}$ again be a pair of machines. We show the procedure in each case for $S_{\text{rest}}(q)$ and x , all of which follows symmetrically for $S_{\text{rest}}(q')$ and x' . Adding jobs from S_{rest} to $S_{8,10}$ or scheduling them directly on q always implies the removal of these jobs from S_{rest} . In some of the following cases, we remove enough load from either $S_{\text{rest}}(q)$ or $S_{\text{rest}}(q')$ such that the process need not be executed for q' or q . If this occurs for either q or q' , we assume wlog. that it always occurs for q , i.e. the machine we consider first, and simply add all of $S_{\text{rest}}(q')$ to S_{nice} . The process as given by the case distinction is only executed for q if $x > 0$, as otherwise, $L(S_{\text{rest}}(q))$ is small enough already for our purposes. We write $\exists i \in S_{\text{rest}}(q)$ iff there are some job pieces of class i in $S_{\text{rest}}(q)$.

Case 4: $\exists i \in S_{\text{rest}}(q) : s_i > x \wedge s_i$ appears more than once in $S_{\text{rest}}(q)$.

Adding $S_{\text{rest}}(q) \setminus s_i$ to S_{nice} is enough, as $L(S_{\text{rest}}(q) \setminus s_i) = (1/2 + x) - s_i \leq 1/2$, and $\alpha_i = 1$, i.e. only one s_i is counted in $L_{\text{nice}}(I_{\text{nice}})$.

Case 5: $\exists i \in S_{\text{rest}}(q) : s_i > x \wedge i$ is not fully scheduled on q .

Case 5.1: $s_i \leq 1/6$.

If $s_i \in S_{\text{nice}}$ already before consideration of this case, we simply add $S_{\text{rest}}(q) \setminus s_i$ to S_{nice} , maintaining $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$, and add i_8, i'_8 to $S_{8,10}$. Thus, in the following assume otherwise. Let $M_{\text{single}}^i \subseteq M_{\text{single}}$ be the subset of machines in M_{single} with job pieces of i scheduled on them. Let $P' = s_i$ before the first machine in M_{single}^i is considered in the case distinction.

For some set of job pieces $J' \in C_i, J' \in S_{\text{rest}}(q)$ with $P(J') = \min(\sum_{j \in C_i, \sigma^{\text{OPT}}(j)=q} p_j, P')$, we add J', i_8 and i'_8 to $S_{8,10}$. If $s_i \notin S_{8,10}$, we additionally add s_i to $S_{8,10}$. If $P(J') < P'$ and $s_i \notin S_{\text{nice}}$, we add $S_{\text{rest}}(q) \setminus J'$ to S_{nice} ; otherwise, we add $S_{\text{rest}}(q) \setminus \{J', s_i\}$ to S_{nice} . We set $P' = P' - P(J')$.

Essentially, we make sure that only one s_i is included in $S_{8,10}$ and S_{nice} , respectively. Before adding s_i to S_{nice} , we also make sure to add at least s_i processing time of jobs in C_i to $S_{8,10}$ instead of S_{nice} . We show that this is enough to keep the overall load added to S_{nice} small enough such that $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$ is preserved. Let p'_j be the processing time of a job j over all machines $q_i \in M_{\text{single}}^i$ in σ^{OPT} . If $\sum_{j \in C_i} p'_j \geq s_i$, the load added to S_{nice} over all machines in M_{single}^i can be expressed as

$$\sum_{q_i \in M_{\text{single}}^i} (L_{\text{rest}}(q_i) - s_i) + s_i - s_i \leq \sum_{q_i \in M_{\text{single}}^i} (5/9 - x) \leq |M_{\text{single}}^i|/2 ,$$

as there is at most one setup time s_i added to S_{nice} over all machines in M_{single}^i , but also at least s_i processing time of $\sum_{j \in C_i} p'_j$ not added to S_{nice} in the process. If $\sum_{j \in C_i} p'_j < s_i$, the load added to S_{nice} over all machines in M_{single} can instead be expressed as $\sum_{q_i \in M_{\text{single}}^i} (L_{\text{rest}}(q_i) - s_i) \leq |M_{\text{single}}^i|/2$, as s_i is always in $S_{\text{rest}}(q_i)$ for all q_i , but never added to S_{nice} . This maintains our goal of adding at most $(L(S_{\text{rest}}(q_i)) - \max(0, x)) \leq 1/2$ load per machine $q_i \in M_{\text{single}}^i$ to S_{nice} .

Now, we look at the average load added to $S_{8,10}$ over all such machines. As $s_i \notin S_{\text{nice}}$ before the beginning of this process, and i is not fully scheduled on q , there are two possibilities: There exists some $q_i \in M_{\text{dual}} \cup M_{10}$ such that i is partly scheduled on q_i in σ^{OPT} , or not. In the former case, at most s_i load is added to $S_{8,10}$ per machine q . In the latter case, there is at most $2s_i$ load added to $S_{8,10}$, but since i is not fully scheduled on q , neither on machines in M_{nice} or $M_{\text{dual}} \cup M_{10}$, it must be scheduled on another machine in M_{single} . Thus, we add at most $s_i \leq 1/6$ load to $S_{8,10}$ per machine. As this added load always includes a setup time s_i , we add at most s_i processing time of i to $S_{8,10}$ here. If, in fact, $s_i > 1/9$, we already reduce the load of S_{rest} by at least $x + x'$ per machine. Thus, in this case, the process is not executed for q' .

Case 5.2: $s_i > 1/6$.

Then all other job pieces of i that are scheduled on some other machines in σ^{OPT} , are scheduled either on other machines in M_{single} , are already included in S_{nice} , or both: Due to the high setup time, such job pieces cannot be scheduled on machines of $M_{\text{dual}} \cup M_{10}$ in σ^{OPT} . If some job piece is already in S_{nice} , then so is s_i , and we simply add $S_{\text{rest}}(q) \setminus s_i$ to S_{nice} , with the load of this added set being at most $5/9 - s_i < 1/2$. If this is not the case, then there are some other job pieces of class i scheduled on at least one other machine in M_{single} . Let $M_{\text{single}}^i \subseteq M_{\text{single}}$ be the set of these machines. We add $S_{\text{rest}}(q_i) \setminus s_i$ to S_{nice} , for all $q_i \in M_{\text{single}}^i$, as well as s_i only once. This implies that the load put into S_{nice} over all $|M_{\text{single}}^i| \geq 2$ of these machines is at most

$$\begin{aligned} s_i + |M_{\text{single}}^i|(5/9 - s_i) &= |M_{\text{single}}^i|/2 + |M_{\text{single}}^i|/18 - (|M_{\text{single}}^i| - 1)s_i \\ &\leq |M_{\text{single}}^i|/2 , \end{aligned}$$

where the last inequality follows from the fact that $1/6 < s_i$ and $|M_{\text{single}}^i| \geq 2$. This maintains our goal of adding at most

$(L(S_{\text{rest}}(q_i)) - \max(0, x)) \leq 1/2$ load per machines $q_i \in M_{\text{single}}^i$ to S_{nice} . There is no load added to $S_{8,10}$ in this case.

Case 6: $\exists i \in S_{\text{rest}}(q) : s_i > x \wedge i$ is fully scheduled on $q \vee \nexists i \in S_{\text{rest}}(q) : s_i > x$.

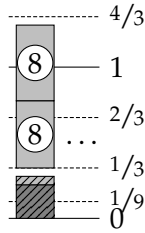
We know that, if such an i exists, that $s_i + P(C_i) < 1/2$, as otherwise, q would have been considered already in [Case 3](#).

Case 6.1: $s_i + P(C_i) \leq 1/6$.

Then also $s_i \leq 1/6$. We add $s_i + P(C_i)$, i. e. at most $1/6$ load, to $S_{8,10}$. If, in fact, $s_i + P(C_i) > 1/9$, we already reduce the load of S_{rest} by at least $x + x'$ per machine, and the process is not executed for q' . Otherwise, $s_i \leq 1/9$. In any case, the load of $S_{\text{rest}}(q) \setminus \{s_i, C_i\}$ has load at most $L(S_{\text{rest}}(q)) - s_i + P(C_i) < L(S_{\text{rest}}(q)) - x$, which we then can simply add to S_{nice} .

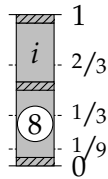
Case 6.2: $1/6 < s_i + P(C_i) \leq 1/3$.

We add $S_{\text{rest}}(q) \setminus \{s_i, C_i\}$ to S_{nice} . If $s_i \leq 1/9$ and $s_i + P(C_i) \leq 2/9$, we add s_i and C_i to $S_{8,10}$. Otherwise, we directly schedule $s_i + P(C_i)$ on q , starting at 0, and both i_8 and i'_8 above, starting at $1/3$. This maintains $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$: The load of S_{nice} is overall increased by at most $L(S_{\text{rest}}) - 1/6 \leq 1$, while q' is added to M_{nice} . In this case, as $s_i + P(C_i) > 1/6 > x + x'$, the process is not executed for q' .



Case 6.3: $1/2 > s_i + P(C_i) > 1/3 \vee \nexists i \in S_{\text{rest}}(q), s_i > x$.

If such an i exists, it is the only class scheduled on q in σ^{OPT} with $s_i > x$, as only one class of processing time $> 1/3$ can be fully scheduled on q besides i_8 . Then i is also scheduled on q without interruption: Otherwise, one would need at least two setups of size $s_i > x$, which would have been handled by [Case 4](#). Therefore, the setup times and job pieces in $S_{\text{rest}} \setminus i$ all belong to some classes with setup times $< x$, and are scheduled below both i_8 and i , between these two, or above both in σ^{OPT} , as pictured below.



Since $s_i + P(C_i) < 1/2$, the total load of $L(S_{\text{rest}}(q)) - (s_i + P(C_i))$ on q is greater than x . We now take at least x load of classes with $s_i \leq x$ from q and add it to $S_{8,10}$, taking only job pieces that are scheduled in the intervals $[8/9 + 2Y, 1]$, $[4/9 + Y, 5/9 - Y]$, and $[0, 1/9 - 2Y]$ on q in σ^{OPT} . We know that, in σ^{OPT} , there is at least x load in each interval of size $2(1/18 - Y)$, as there is at most $1 - (4/9 + Y) - (1/2 + x) = 1/18 - Y - x \leq 1/18 - Y$ free space on each machine and $x \leq 1/18 - Y$ by definition. In fact, over these three intervals together, there is at least x load of classes with setup times $\leq x$ scheduled in σ^{OPT} : The only classes on q with setup time $> x$ are i_8 and i , if i exists. As they are both scheduled on q without interruption, and their combined load is less than $1 - x$, no matter where they are scheduled on q , i.e. either one at the top and at the bottom or both in the middle, there is always at least x load of classes with setup time $\leq x$ over the three intervals. To achieve this, we execute the following procedure:

```

1: procedure REMOVE_TINY_JOBS( $x, q$ )
2:    $ret = \emptyset$ 
3:   for all  $\mathcal{I} \in [8/9 + 2Y, 1], [4/9 + Y, 5/9 - Y], [0, 1/9 - 2Y]$  do
4:     if  $x > 0$  then
5:       Let  $C'_x := \{C'_1, \dots, C'_c\}$ , where  $C'_k$  is defined to be
       the set of job parts of jobs in  $C_k$  that are scheduled in  $\mathcal{I}$  on
        $q$  if  $s_k \leq x$ , and  $C'_k = \emptyset$ , else, for all  $k \in [c]$ . For simplicity
       reasons, here, let  $s_k := 0$  if  $C'_k = \emptyset$ .
6:       if  $\sum_{k \in C'_x} (s_k + P(C'_k)) \leq x$  then
7:          $ret = ret \cup \bigcup_{k \in C'_x} s_k \cup C'_x$ 
8:          $x = x - \sum_{k \in C'_x} (s_k + P(C'_k))$ 
9:       else
10:         $k = 1$ 
11:        while  $x > 0$  do
12:          if  $s_k + P(C'_k) \leq 2x$  then
13:             $ret = ret \cup s_k \cup C'_k$ 
14:             $x = x - s_k - P(C'_k)$ 
15:             $k = k + 1$ 
16:          else
17:            Let  $J^x \subseteq C'_k$  be a subset of job pieces
            with proc. time exactly  $x$ .  $J^x$  always exists in  $\mathcal{I}$  due to the
            fact that  $s_k \leq x$  and  $x \leq 1/18 - Y$ .
18:             $ret = ret \cup s_k \cup J^x$ 
19:             $x = x - s_k - P(J^x)$ 
20:          end if
21:        end while
22:      end if
23:    end if
24:  end for return  $ret$ 

```

25: **end procedure**

By definition of this procedure, ret only included jobs from the three intervals, and $x \leq L(ret) \leq 2x$. Thus, we add at most $2x \leq 1/9$ load to $S_{8,10}$, while reducing the load of $S_{rest}(q)$ by at least x , making it safe to add the remainder of $S_{rest}(q)$ to S_{nice} while preserving $L_{nice}(I_{nice}) \leq |M_{nice}|$.

After finding the appropriate cases above for all machine pairs $q, q' \in M_{single}$, only the setup times and job pieces in $S_{8,10}$ are not yet scheduled on machines with makespan at most $4/3$. We now aim to schedule these on the machines in $M_{8,10}$ with makespan at most $4/3$, thus obtaining a $4/3$ -approximation for the whole instance. First note that, by construction, there is at least one machine in $M_{8,10}$ for each class of ⑩ in $S_{8,10}$, and at least one machine for each two classes of ⑧ in this set, as well as one machine for classes i_8 and i added to $S_{8,10}$ in [Case 3](#). We start by scheduling each such class of ⑩ on one machine of $M_{8,10}$ starting at $1/3$, and each pair of such ⑧ classes as well as i_8 and i on one machine of $M_{8,10}$ one after another, also starting at $1/3$. We assume that now, each machine in $M_{8,10}$ either has one ⑩ class, two ⑧ classes, or i_8 and i scheduled on it, always starting at $1/3$.

Afterwards, all non scheduled job pieces in $S_{8,10}$ belong to classes $i \in \textcircled{7}$. We now want to place these job pieces on the machines of $M_{8,10}$, between 0 and $1/3$. If we manage to do this in a feasible way, we are done: All other classes of ⑦ are part of I_{nice} , where all job pieces of ⑦ classes are scheduled above $1/3$ by definition of the algorithm given in [Theorem 12](#). Note that the only times setup times and job pieces are added to $S_{8,10}$ are at the start of this section, where all setup times and job pieces machines in $M_{dual} \cup M_{10}$ are added to $S_{8,10}$, and in cases [Case 3](#), [Case 5.1](#), and [Case 6](#). As the total load of setup times and processing times that are added to $S_{8,10}$ is at most $2/9$ per machine added to $M_{8,10}$, it holds that $L(S_{8,10}) \leq 2/9|M_{8,10}|$.

To schedule the job pieces in $S_{8,10}$, we use WRAPLJ. For this, we construct a wrap sequence Q from $S_{8,10}$ and a respective wrap template ω on the machines in $M_{8,10}$. To show feasibility of WRAPLJ for Q and ω , we prove that the prerequisites of both [Property 3](#) and [Property 5](#) are fulfilled.

Let $Q := (s_{i_1}, j_1^1, \dots, j_{n_1}^1, s_{i_2}, j_1^2, \dots, j_{n_2}^2, \dots, s_{i_k}, j_1^k, \dots, j_{n_k}^k)$ be the wrap sequence constructed by taking all job pieces in $S_{8,10}$ grouped by their class, and adding the setup time of their respective class to the sequence, and let $\omega = (\omega_1, \dots, \omega_{|M_{8,10}|})$, with $\omega_r = (r, 0, 1/3)$, $\forall 1 \leq r \leq |M_{8,10}|$. Since there is at least one setup time per class of every job piece in $S_{8,10}$ by construction of the set, and only one per such class in Q , it holds that $L(Q) \leq L(S_{8,10})$.

We first show that the requirement of [Property 3](#) is fulfilled. Since $b_r - a_r = 1/3 > \max_{r \in [\omega]} s_r$ is fulfilled for all $r \in [\omega]$ by virtue of all job pieces in Q being part of ⑦ classes, it is only left to show that $L(Q) \leq S(\omega) - \sum_{r=2}^{|\omega|} s_r$. First note that, for all job pieces $j \in S_{8,10}$ with

$s_j > 1/9$, it holds that all such j are job pieces that were previously scheduled on machines in M_{10} , or that were added in [Case 5.1](#) or [Case 6.1](#): In all other cases, all job pieces added to $S_{8,10}$ belong to classes i with $s_i \leq 1/9$. Simultaneously, the maximum setup time s_j of such j , as well as the maximum load added to $S_{8,10}$ per machine, is $1/6$, either by the definition of [⑩](#), as shown in [Case 5.1](#), or by definition of [Case 6.1](#). In all other cases, i.e. for all $j \in S_{8,10}$ with $s_j \leq 1/9$, the maximum load added to $S_{8,10}$ per machine is $2/9$. We again define p'_j to be total processing time of all job pieces of a job j included in $S_{8,10}$. Let $L_1 = \sum_{j \in S_{8,10}, s_j > 1/9} p'_j + \sum_{i \in S_{8,10}, s_i > 1/9} s_i$ and $L_2 = \sum_{j \in S_{8,10}, s_j \leq 1/9} p'_j + \sum_{i \in S_{8,10}, s_i \leq 1/9} s_i$. By this definition, $L_1 + L_2 = L(Q)$. Then, by the maximum load added to $S_{8,10}$ per machine in $M_{8,10}$, there exists some $r' \in |\omega|$ with $1/6r' + 2/9(|\omega| - r') \geq \lceil L_1 + L_2 \rceil$, such that L_1 can be fully scheduled on machines $1, \dots, r'$, and L_2 can be fully scheduled on machines $r', \dots, |\omega|$. This implies that $s_r \leq 1/6$, for the additional setup times s_r on machines $1, \dots, r'$, and $s_r \leq 1/9$ for the additional setup times s_r on machines $r' + 1, \dots, |\omega|$ introduced by the execution of WRAPLJ. Thus,

$$\begin{aligned}
S(\omega) - \sum_{r=2}^{|\omega|} s_r &\geq 1/3|\omega| - \sum_{r=2}^{r'} 1/6 - \sum_{r=r'+1}^{|\omega|} 1/9 \\
&= 1/3r' + 1/3(|\omega| - r') - 1/6r' - 1/9(|\omega| - r') \\
&\geq 1/6(r' - 1) + 2/9(|\omega| - r') \geq L_1 + L_2 - 1/6 \\
&= L(Q) - 1/6,
\end{aligned}$$

and the claim follows.

We now show that the requirement of [Property 5](#) is fulfilled. It holds for all ω_r, ω_{r+1} that $u_r < u_{r+1}$, $b_r = b_{r+1}$, and $a_r = a_{r+1}$, for all $r \in [1, |\omega| - 1]$. Thus, it is only left to show that, for all $r \in [|\omega|]$, $\max_{\ell \in [k], y \in n_\ell} s_{i_\ell} + j_y^\ell \leq b_r - a_r$. To prove this, as $b_r - a_r = 1/3$ for all r , it is enough to show that for every job $j \in S_{8,10}$, $s_j + p'_j \leq 1/3$, where p'_j is the processing time of j included in $S_{8,10}$. We first look at all jobs $j \in S_{8,10}$ that belong to some class i with $s_i > 1/9$. As argued before, all such jobs were previously scheduled on machines in M_{10} in σ^{OPT} , or were added in [Case 5.1](#) or [Case 6.1](#). In [Case 6.1](#), only full classes i with $s_i + P(C_i) \leq 1/6$ are added to $S_{8,10}$. Thus, no job piece of a job of this class is added in any other step, and $s_i + p'_j \leq 1/6$. There is at most $1/3 - 2s_i$ processing time of a single job j of C_i on machines in M_{10} by definition of [⑩](#). As there is at most s_i processing time of jobs in C_i to $S_{8,10}$ in [Case 5.1](#), it holds for each such job j that $s_i + p'_j \leq s_i + 1/3 - 2s_i + s_i \leq 1/3$.

Now, consider all jobs $j \in S_{8,10}$ that belong to some class i with $s_i \leq 1/9$. Pieces of such jobs may have been previously scheduled on machines of $M_{\text{dual}} \cup M_{10}$ in σ^{OPT} , or added to $S_{8,10}$ in cases [Case 3](#), [Case 5.1](#), or [Case 6](#). First, we look at jobs added in [Case 6.1](#) or [Case 6.2](#). Any job piece added in either of these cases belongs to a class i that

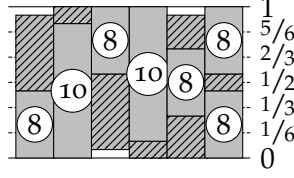


Figure 3.3: The position of ⑧ and ⑩ classes in σ^{OPT} .

was fully scheduled on a machine $q \in M_{\text{single}}$ in σ^{OPT} . This implies that no job piece of this class was scheduled on any other machine in σ^{OPT} . Thus, $s_i + p'_j \leq 1/3$ by definition of [Case 6.1](#) and [Case 6.2](#).

We now analyze the total processing time of job pieces of j of classes i with $s_i \leq 1/9$ in $S_{8,10}$ that have been added to this set in any or all of the other cases. For this, we look at the total size of the intervals where job pieces are taken from σ^{OPT} and added to $S_{8,10}$. As σ^{OPT} is a feasible solution with no parallelization, the total processing time of j added to $S_{8,10}$ over all these intervals in all these cases is at most the total non-overlapping size of these intervals minus one required setup time per interval. On machines of M_{10} , pieces of j can only be scheduled in the intervals $[8/9 - Y, 1]$ and $[0, 1/9 + Y]$ by definition of Y in σ^{OPT} . On machines of M_{dual} , as both i_8 and i'_8 are scheduled without interruption, j can only be scheduled in the intervals $[8/9 + 2Y, 1]$, $[4/9 + Y, 5/9 - Y]$ and $[0, 1/9 - 2Y]$. Similarly, for machines considered in [Case 3](#), pieces of j can only be scheduled in intervals $[8/9 + Y, 1]$, $[4/9 + Y, 5/9 - Y]$ and $[0, 1/9 - Y]$. By definition of [Case 6.3](#), all job pieces of j in $S_{8,10}$ were scheduled only in the intervals $[8/9 + 2Y, 1]$, $[4/9 + Y, 5/9 - Y]$, and $[0, 1/9 - 2Y]$. This situation is visualized in [Figure 3.3](#).

The total non-overlapping size of these intervals, minus one required setup time s_i per interval, then is

$$2(1/9 + Y - s_i) + (1/9 - 2Y - s_i) = 1/3 - 3s_i$$

The only time a job piece of j may have been added to $S_{8,10}$ outside these intervals is in [Case 5.1](#). The total processing time of a single class, and thus also a single job, added to $S_{8,10}$ in [Case 5.1](#) is at most s_i . Thus, $p'_j \leq 1/3 - 2s_i$ and $s_i + p'_j \leq 1/3 - s_i$.

The only thing left to do now is to deal with $I_{8\text{rem}}$. In total, we have added setup times, job pieces and machines to $I_{8\text{rem}}$ at most three times:

1. When pairing machine of $M_{\text{single}} \cup M_{\text{dual}}$ which have some part of a nice class in ①, ..., ⑥ scheduled on them, where we add no machines and a class $i_8^1 \in \text{⑧}$ to $I_{8\text{rem}}$.
2. When pairing machines of M_{single} that have class $i_{5,6} \in \text{⑤} \cup \text{⑥}$ fully scheduled on them, where we add one machine q_2 , a class $i_8^2 \in \text{⑧}$ and $S_{\text{rest}}(q_2)$ to $I_{8\text{rem}}$.

3. When pairing the rest of the machines of M_{single} , where we add one machine q_3 , a class $i_8^3 \in \textcircled{8}$, and $S_{\text{rest}}(q_3)$ to $I_{8\text{rem}}$.

We also know that there is currently no class of $\textcircled{8}$ in S_{nice} by all previous steps of our algorithm, but a nice instance allows for at most one class of $\textcircled{8}$ by Definition 9. The number of machines in $I_{8\text{rem}}$ is thus either 0, 1, or 2.

If the number of machines is 0 or 2, neither or both $q_2 \in I_{8\text{rem}}$ and $q_3 \in I_{8\text{rem}}$. In the latter case, we schedule i_8^2 and i_8^3 together on q_2 , starting at $1/3$. Then, the sub-instance I' induced by $S_{\text{rest}}(q_2) \cup S_{\text{rest}}(q_3)$ and q_3 has $m_{\text{nice}}(I') = 1$. As $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$, $m_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$ by Lemma 10. Thus, $m_{\text{nice}}(I_{\text{nice}} \cup I') \leq |M_{\text{nice}}| + 1$ by definition of m_{nice} , and Theorem 12, (ii) holds for I_{nice} after adding $S_{\text{rest}}(q_2) \cup S_{\text{rest}}(q_3)$ to I_{nice} and q_2 to M_{nice} . Now, either there are no more setup times and job pieces in $I_{8\text{rem}}$, and we are done, or only i_8^1 is in $I_{8\text{rem}}$. Since, when adding the machine q_1 on which i_8^1 was scheduled in σ^{OPT} , we have increased $|M_{\text{nice}}|$ by one, but added only at most $1 - L(i_8^1) - 1/3 \geq 2/9$ load to S_{nice} , implying that $7/9$ load can be added to S_{nice} without violating $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$. We can thus simply add i_8^1 to S_{nice} as the single class of $\textcircled{8}$ in I_{nice} .

If the number of machines is 1, either machine $q_2 \in I_{8\text{rem}}$ or $q_3 \in I_{8\text{rem}}$. If now $i_8^1 \notin I_{8\text{rem}}$, we simply add $q \in \{q_2, q_3\}$ to M_{nice} , and all setup times and job pieces that were scheduled on q in σ^{OPT} to S_{nice} , keeping the instance nice with exactly one class of $\textcircled{8}$. If instead $i_8^1 \in I_{8\text{rem}}$, we schedule both classes of $\textcircled{8}$ in $I_{8\text{rem}}$ on q , and add all setup times and job pieces previously scheduled on q but the $\textcircled{8}$ class to S_{nice} . As the total load of these setup times and job pieces is at most $5/9$, but we can add $7/9$ to S_{nice} without violating $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$ if $i_8^1 \in I_{8\text{rem}}$, the resulting schedule is feasible.

The resulting schedule is now a feasible $4/3$ -approximation of the original instance. This then, in total, shows that for every optimal solution σ^{OPT} of an instance with makespan T and more classes of $\textcircled{8}$ than $\textcircled{9}$, there exists a feasible schedule with makespan at most $4/3T$, where

- all classes of $\{\textcircled{1}, \dots, \textcircled{6}\}$ and some partial classes of $\textcircled{7}$ are part of a nice instance
- all classes of $\textcircled{9}$ are scheduled together with classes of $\textcircled{8}$ with no other setup times or job pieces on one machine each
- all pairs of classes of $\textcircled{8}$ and $\textcircled{10}$ with combined load less than $4/3T$ are scheduled together with no other setup times or job pieces on one machine each
- all other but at most two $\textcircled{8}$ classes are scheduled in pairs on a machine each with starting time $1/3T$
- of those at most two $\textcircled{8}$ classes, one is included in the nice instance, and the other is scheduled together with a $\textcircled{7}$ class i

with load $5/9T \geq s_i + P(C_i) \geq 1/2T$ on one machine with starting time $1/3T$

- all other ⑩ classes are scheduled on a machine each with starting time $1/3T$
- all remaining ⑦ classes i not included in the nice instance with load $5/9T \geq s_i + P(C_i) \geq 1/2T$ are scheduled in pairs on a machine each
- all other remaining ⑦ classes i have $s_i \leq 1/6T$ and are scheduled below $1/3T$ together with classes of ⑧ and ⑩ above, with at most $2/9T$ processing time per machine for all classes such ⑦ classes i with $s_i \leq 1/9T$, and at most $1/6T$ processing time per machine for all such ⑦ classes i' with $1/6T \geq s_{i'} > 1/9T$

3.5.5 Classes of Type Nine Remain

Now, we consider the case where, after our matching of ⑧ and ⑨ classes, the only machines with non-nice types of classes not yet modified contain full classes of ⑨ and ⑩. Let $M_{9,10}$ be the subset of machines with these classes scheduled on them in σ^{OPT} . For each job $j \in \mathcal{J}$, let p'_j be its processing time scheduled on machines of $M_{9,10}$ in σ^{OPT} , and $p''_j = p_j - p'_j$. For each class i , let $P'(C_i) = \sum_{j \in C_i} p'_j$. We can make the following statement about the number of setup times s_i per class i on $M_{9,10}$ in σ^{OPT} .

Lemma 11. *The number λ'_i of setup times s_i on $M_{9,10}$ in σ^{OPT} is at least $\lceil P'(C_i)/(1/3 - s_i) \rceil$.*

Proof. By definition of the machines in $M_{9,10}$, at most $1/3 - s_i$ processing time of job pieces of any class i can be scheduled on each machine in σ^{OPT} . Thus, $P'(C_i)$ is processed on at least $\lceil P'(C_i)/(1/3 - s_i) \rceil$ different machines of $M_{9,10}$ in σ^{OPT} , requiring at least one setup time per machine. $\square$

This statement also implies that there is at least one setup s_i per job j of class i with $s_i + p'_j > 1/3$ on the machines in $M_{9,10}$. We start by removing all job pieces scheduled on machines of $M_{9,10}$ in σ^{OPT} , putting them into a set $S_{9,10}$. As all jobs of classes in ⑦ on machines in $M_{9,10}$ are scheduled in the intervals $[0, 1/3]$ and $[2/3, 1]$, there is at most $2/3 - 2s_i$ processing time of a single job j of class i scheduled on machines in $M_{9,10}$, and every job j with $s_i + p'_j > 1/3$ is scheduled on at least two different machines, and has job pieces in both intervals $[0, 1/3]$ and $[2/3, 1]$ in σ^{OPT} .

Case 1: $\exists j \in S_{9,10} : 2s_i + p_j \geq 2/3 \wedge s_i + p'_j > 1/3$.

Note that, for each such j , $p_j'' > 0$, and therefore there is a setup time s_i of its class i already included in I_{nice} . Let $J \subseteq S_{9,10}$ be the set of all such jobs. For each class i , let $P^{\text{over}}(C_i) = \sum_{j \in C_i \setminus J} p_j' - (\lceil 1/3 - s_i \rceil - 2|C_i \cap J|)$, i.e. the total processing time of job pieces of class i in $S_{9,10}$ that have no setup time in $S_{9,10}$ if one would place all jobs of $C_i \cap J$ with exactly two setup times s_i . By Lemma 11, at least $P^{\text{over}}(C_i)$ processing time of job pieces of class i then must be placed together with jobs in $C_i \cap J$ on machines $M_{9,10}$ in σ^{OPT} .

We start by ordering the machines q in $M_{9,10}$ by non-increasing $F_q = 1 - L(i_{9,10})$, where $i_{9,10}$ is the class of $\textcircled{9} \cup \textcircled{10}$ scheduled on q in σ^{OPT} . Let $M'_{9,10}$ be the set of the first $2|J|$ machines by this ordering, and $F = \sum_{q \in M'_{9,10}} F_q$. Then, $\sum_{j \in J} (2s_j + p_j') + \sum_{i \in [c]} P^{\text{over}}(C_i) \leq F$: Let q, q' be the last two machines added to $M'_{9,10}$. By the ordering of the machines, the largest non-overlapping interval not filled with classes of $\textcircled{9} \cup \textcircled{10}$ over all machines in $M_{9,10} \setminus M'_{9,10}$ has size at most $F_q + F_{q'}$. Let $F' = F - \frac{|M'_{9,10}|}{2}(F_q + F_{q'})$ be the total free space of machines $M'_{9,10}$ outside the interval $F_q + F_{q'}$. At most $|J|(F_q + F_{q'})$ total load of $\sum_{j \in J} (2s_j + p_j') + \sum_{i \in [c]} P^{\text{over}}(C_i)$ can be placed in any non-overlapping interval of size $(F_q + F_{q'})$ over all machines in $M_{9,10}$, and the rest must be placed in F' , as otherwise, either some job $j \in J$ would be parallelized in σ^{OPT} , or there would be more than λ_i' setup times for some class i in σ^{OPT} . This implies that $\sum_{j \in J} (2s_j + p_j') + \sum_{i \in [c]} P^{\text{over}}(C_i) \leq F' + |J|(F_q + F_{q'})$. As $F = F' + |M'_{9,10}|/2(F_q + F_{q'}) = F' + |J|(F_q + F_{q'})$, the claim that $\sum_{j \in J} (2s_j + p_j') + \sum_{i \in [c]} P^{\text{over}}(C_i) \leq F$ follows.

We order the jobs in J by non-decreasing setup time s_i of their respective classes i . While for the first $j \in J$, $F_q + F_{q'} \geq s_i + 1/3$ for the first two machines $q, q' \in M'_{9,10}$, we know that, using the extra $1/3$ space given per machine by only approximating the optimal solution, the total free space on these two machines is at least $2/3 + F_q + F_{q'} \geq 2/3 + s_i + 1/3 = s_i + 1$. As it holds for all j that $p_j \leq 1 - s_i$ by definition of σ^{OPT} , we can schedule the whole job j on q and q' together with $i_{9,10}$ and $i'_{9,10}$, the classes of $\textcircled{9} \cup \textcircled{10}$ scheduled on q and q' in σ^{OPT} . To do this without parallelization of j , we first schedule $i_{9,10}$ on q such that it finishes at exactly $4/3$, $i'_{9,10}$ on q' such that it starts at 0, proceeding with j by scheduling s_i as well as at most $4/3 - L(i_{9,10}) - s_i$ processing time of j on q starting at 0, and s_i as well as $p_j - (4/3 - L(i_{9,10}) - s_i)$ processing time of j on q' , starting at $L(i'_{9,10})$. This removes p_j'' processing time of j from S_{nice} . Since $2s_i + p_j \geq 2/3$, the total load of setup times and job pieces on q and q' is at least 2 after this process.

Afterwards, we add such placed j to a set J_1 , and q, q' to a set M_{large} . Let $J_2 = J \setminus J_1$ be the set of jobs of J not placed by

this process. It now holds, by the ordering of J and $M'_{9,10}$, that $F_q + F_{q'} < s_j + 1/3$, for all $j \in J$ and $q, q' \in M'_{9,10} \setminus M_{\text{large}}$. As all jobs of J_1 have been scheduled on exactly two machines of $M'_{9,10}$, we know that $M'_{9,10} \setminus M_{\text{large}} = 2|J_2|$. Thus, $\sum_{j \in J_2} s_j + 1/3 > \sum_{q \in M'_{9,10} \setminus M_{\text{large}}} F_q$.

Let $P^{\text{ext}} = \sum_{j \in J_1} p_j''$. As now $L(q) > 1$ for each $q \in M_{\text{large}}$,

$$\begin{aligned}
& P^{\text{ext}} + \sum_{j \in J_1} (2s_j + p_j') + \sum_{j \in J_2} (s_j + 1/3) \\
& > \sum_{q \in M_{\text{large}}} F_q + \sum_{q \in M'_{9,10} \setminus M_{\text{large}}} F_q = F \\
& \geq \sum_{j \in J} (2s_j + p_j') + \sum_{i \in [c]} P^{\text{over}}(C_i) \\
& \Leftrightarrow P^{\text{ext}} > \sum_{j \in J_2} (s_j + p_j' - 1/3) + \sum_{i \in [c]} P^{\text{over}}(C_i)
\end{aligned}$$

As exactly P^{ext} load is removed from S_{nice} when scheduling the jobs from J_1 , we can safely move $s_j + p_j' - 1/3$ processing time of each $j \in J_2$, as well as $P^{\text{over}}(C_i)$ processing time of some jobs $j \in S_{9,10}$ of each class i , from $S_{9,10}$ to S_{nice} without violating $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$. Then, the remaining processing time of each $j \in J_2$ is at most $p_j' - (s_j + p_j' - 1/3) \leq 1/3 - s_j$. Now, all jobs j with previously $s_i + p_j' > 1/3 \wedge 2s_i + p_j \geq 2/3$ have either been fully placed on machines in M_{large} (the jobs in J_1), or it now holds for such jobs that $s_j + p_j' \leq 1/3$ (the jobs in J_2), and there are thus no more jobs with $s_j + p_j' > 1/3$ to consider. Additionally, for all $i \in [c]$, by this we ensure $P^{\text{over}}(C_i) \leq 0$, i. e. there are at least $\left\lceil \sum_{j \in C_i} p_j' / (1/3 - s_i) \right\rceil$ setup times left in $S_{9,10}$ for each class i .

Case 2: $\exists j \in S_{9,10} : 2s_i + p_j > 2/3 \wedge s_i + p_j' \leq 1/3 \wedge M_{9,10} \neq \emptyset$.

We remove machine the q with largest $F_q = 1 - L(i_{9,10})$ from $M_{9,10}$, where $i_{9,10}$ is the class of $\textcircled{9} \cup \textcircled{10}$ scheduled on a machine in σ^{OPT} . We then schedule the class $i_{9,10}$ on q starting at $1/3$, and schedule s_i as well as $1/3 - s_i$ processing time of j directly below on q , starting at 0. To ensure $\left\lceil \sum_{j \in C_i} p_j' / (1/3 - s_i) \right\rceil$ setup times are left for i in $S_{9,10}$ afterwards, we remove $1/3 - s_i - p_j'$ processing time of some jobs j' , with $j' \in C_i$, from $S_{9,10}$ if possible, and add this processing time as a new job piece to S_{nice} . As for such j , $p_j'' > 0$, there is a setup time s_i already included in S_{nice} , and we have just scheduled $1/3 - s_i - p_j'$ processing time from S_{nice} on q , this preserves $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$ as well.

Afterwards, it holds for all $j \in S_{9,10}$ that $2s_i + p_j \leq 2/3$. On all machines $1, \dots, |M_{9,10}| \in M_{9,10}$, we schedule all remaining classes $i_{9,10} \in \textcircled{9} \cup \textcircled{10}$

alternately starting at 0 or finishing at $4/3$, such that on machine $|M_{9,10}|$, the last class $i_{9,10}$ finishes at $4/3$. This schedules all remaining classes in $\textcircled{9} \cup \textcircled{10}$ by definition of $M_{9,10}$. We define a Wrap Template $\omega = (\omega_1, \dots, \omega_{|M_{9,10}|})$, with $a_r = 0$ and $b_r = 4/3 - L(i_{9,10})$ if $i_{9,10}$ finishes at $4/3$ on machine r , or with $a_r = 4/3 - L(i_{9,10})$ and $b_r = 4/3$ if $i_{9,10}$ starts at 0 on machine r . Let $F = \sum_{r=1}^{|\omega|} (b_r - a_r - 1/3)$ be the total space on machines $1, \dots, |\omega|$ available in σ^{OPT} besides the classes of $\textcircled{9} \cup \textcircled{10}$ and the $1/3$ space giving by the approximation error. Let Q be an empty Wrap Sequence. We first execute [Algorithm 6](#) to fill Q . We then execute

Algorithm 6

```

1: while  $S_{9,10} \neq \emptyset$  and  $F > 0$  do
2:   Let  $i$  be a class with  $s_i \in S_{9,10}$ .
3:   Move  $s_i$  from  $S_{9,10}$  to  $Q$  and set  $F := F - s_i$ .
4:   while  $F > 0$  and  $\exists j : j \in C_i \wedge p'_j > 0$  do
5:     Remove  $p'_j$  from  $S_{9,10}$ ,  $p''_j$  from  $S_{\text{nice}}$ .
6:     if  $F > p_j$  then
7:       Add  $j$  to  $Q$  and set  $F = F - p_j$ .
8:     else
9:       Split  $p_j$  into two parts  $j^1$  and  $j^2$  with  $p_{j^1} = F$  and  $p_{j^2} = p_j - F$ .
10:      Add  $j^1$  to  $Q$ ,  $j^2$  to  $S_{\text{nice}}$ , and set  $F := 0$ .
11:     end if
12:   end while
13: end while

```

$\text{WRAPLJ}(Q, \omega)$. [Property 3](#) is fulfilled: $L(Q) \leq S(\omega) - \sum_{r=2}^{|\omega|} s_r$ is true as $\sum_{r=2}^{|\omega|} s_r \leq (|\omega| - 1)1/3$ by definition of classes in $S_{9,10}$, $L(Q) = F$ by definition of the procedure above, and $S(\omega) = F + (|\omega| - 1)1/3$ by definition of ω . The expression $b_r - a_r \geq 1/3 \geq \max_{r \in [|\omega|]} s_r$ also holds by definition of $\textcircled{9}$. The schedule produced by $\text{WRAPLJ}(Q, \omega)$ is also never parallelized, as only jobs j of classes i with $2s_i + p_j \leq 2/3$ are placed, and, by our definition of ω , no matter where a job begins, the next $2/3$ space in ω does not overlap. Note that, for every class i with $s_i + P(C_i) > 1/3$ and $p'_j > 0$, there are at least two setup times scheduled in σ^{OPT} : Either two s_i are needed to schedule i fully on machines in $M_{9,10}$, or i is only partly scheduled on machines in $M_{9,10}$, and thus, s_i is already included also in S_{nice} . As by definition of WRAPLJ , we only use one *real* s_i for each such class when wrapping it, as the total sum of all additional setup times is lesser or equal than the additional space, we can add one s_i for each such i in S_{nice} without increasing the total load to be scheduled.

If, after this process, $S_{9,10} = \emptyset$, we are done: All jobs previously (partly) scheduled on machines in $M_{9,10}$ are either fully scheduled on these machines, or below $1/3$ if they had less than $1/3$ processing time placed on machines in $M_{9,10}$ in σ^{OPT} ([Case 2](#)). The resulting schedule is

feasible and has makespan at most $4/3$ by construction. Simultaneously, we have ensured at every step that $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$, and we are thus able to compute a $4/3$ -approximation for I_{nice} in time $\mathcal{O}(n)$ by [Theorem 12](#). As all jobs of classes in $\textcircled{7}$ that may have job pieces in both $S_{9,10}$ and S_{nice} are jobs considered in [Case 2](#), and the pieces in $S_{9,10}$ are feasibly placed below, the pieces in S_{nice} feasibly placed above $1/3$, the resulting schedule has no parallelization, and is thus a feasible $4/3$ -approximation.

Let us thus consider the case that $S_{9,10} \neq \emptyset$ after the process. We add $S_{9,10}$ to S_{nice} , removing all duplicate setup times from S_{nice} . Then there is at most one class e with a higher number of setup times in our current solution compared to the original solution σ^{OPT} : For all classes i that are fully scheduled on machines in $M_{9,10}$, we use at most λ'_i setup times, by ensuring in [Case 1](#) and [Case 2](#) that there always are enough setup times left afterwards to schedule the remaining job pieces of each class, and only including jobs of classes i in Q which have at least one setup time s_i still left in $S_{9,10}$ after these cases. All classes i that already had some job pieces in S_{nice} before $S_{9,10}$ was added to S_{nice} , or that are now fully included in S_{nice} , have exactly one setup time in S_{nice} . Only at most a class e , the class which jobs are added last to Q in [Algorithm 6](#), may now have an additional setup time in S_{nice} if e was partly added to Q and partly left in $S_{9,10}$. However, this setup time is only additional iff $s_e + P(C_e) \leq 1/3$ by [Lemma 11](#). Thus, if e now actually has more setup times than in σ^{OPT} , we can conclude that $s_e + P(C_e) \leq 1/3$. As there is at least $1/3$ free space on machine ω_1 by definition of ω , scheduling e fully on this machine and removing s_e from S_{nice} implies that the total number of setup times of each class in our current solution is lesser or equal than this number of each class in σ^{OPT} . This implies that the total load of $\textcircled{7}$ classes, and by extension of all classes of all types, in our current solution is lesser or equal than the total load of these classes in σ^{OPT} . Since $L(Q) = F$, it then must hold that $L_{\text{nice}}(S_{\text{nice}}) \leq |M_{\text{nice}}|$, as otherwise, the total load of all setup times and job pieces placed in σ^{OPT} is greater than m , a contradiction to the optimality of the solution. Now, we only have to make sure that can not occur any parallelization for all classes. This is already fulfilled for all classes but e : Besides at most one job $j \in C_e$, there are only full classes and full jobs on machines in $M_{9,10}$; only j can have a job piece in both Q and S_{nice} which can possible overlap. This can only happen if $s_e + P(C_e) > 1/3$, as otherwise, we have already included e fully in Q . Let p'_j be the processing time of all job pieces of j in Q , p''_j the processing time of all job pieces of j in S_{nice} . We slightly modify ω by setting $a_1 = 0$, i. e. increasing the total space of ω by $1/3$. As $2s_i + p_j \leq 2/3$, either $p'_j \leq 1/3$ or $p''_j \leq 1/3$ must hold. If $p'_j \leq 1/3$, we can then simply add j fully to Q without violating any of the Batch Wrapping properties. If instead $p''_j \leq 1/3$, we modify ω again by setting $b_{|\omega|} = 1/3$. By setting $a_1 = 0$ before, it is still guaranteed

that $S(\omega) \geq F$, but now also makes sure that all of p'_j is scheduled in the interval $[0, 1/3]$ on machine $|\omega|$; as [Theorem 12](#) places all jobs of ⑦ classes above $1/3$, this guarantees that no parallelization occurs. We are thus able to compute a feasible $4/3$ -approximation for I_{nice} via [Theorem 12](#), and thus, a $4/3$ -approximation for the whole instance. The resulting schedule, in either case, has the following properties:

- all classes of $\{\textcircled{1}, \dots, \textcircled{6}\}$ and some parts of classes of ⑦ are part of a nice instance
- all classes of ⑧ are scheduled together with classes of ⑨ with no other setup times or job pieces on one machine each
- all other ⑨ classes and all ⑩ classes are scheduled on a machine each
- all remaining jobs j of ⑦ classes are either fully scheduled on machines containing one ⑨ or ⑩ class each, respectively, or scheduled partly on one machine containing a ⑨ or ⑩ class below $1/3T$ iff $2s_i + p_j > 2/3T$

3.6 THE APPROXIMATION ALGORITHM

In this section, we formally describe the $(4/3 + \varepsilon)$ -approximation algorithm. Due to preemption, the optimal makespan OPT is not necessarily integral. One can thus not simply guess all integral values between some lower and upper bound on OPT via binary search. Instead, we allow a multiplicative error of ε on OPT , for any $\varepsilon > 0$, i. e. guess some value $\text{OPT} \leq T \leq (1 + \varepsilon) \text{OPT}$. Any $4/3T$ -approximate solution then is a $(4/3 + \varepsilon)$ -approximate solution w.r.t the optimal makespan OPT . We do this by computing a 2-approximation [41] in time $\mathcal{O}(n)$ with some makespan $\text{OPT} \leq T' \leq 2\text{OPT}$. We set a lower bound $\ell = T'/2$, and an upper bound $u = 2T'$. Then, using Binary Search, we need at most $\mathcal{O}(\log(1/\varepsilon))$ iterations to find some lower bound ℓ' and upper bound u' with $(1 + \varepsilon)\ell' \geq u' \geq \ell'$, such that no solution with makespan ℓ' exists, while we can compute a solution with makespan at most $4/3u'$, implying that $u' \leq (1 + \varepsilon) \text{OPT}$.

We now give a description of the $4/3T$ -approximation algorithm for general instances, for some fixed makespan guess T , which we run for each guess of the binary search. If there exists an optimal solution with makespan T , our algorithm will compute a $4/3T$ -approximate solution, which we call σ ; otherwise, it will allow us to conclude that no such solution exists. Again, we normalize the instance I by dividing all input parameters, as well as T itself, by T . Given an instance I of $\text{P|pmnt, setup} = s_i | C_{\max}$, we structure I into types $\{\textcircled{1}, \dots, \textcircled{10}\}$ as defined in [Table 3.1](#), in time $\mathcal{O}(c)$. Let $\sigma^{4/3}$ be the $4/3$ -approximate solution with properties as described in [Section 3.5](#), following either [Section 3.5.4](#) or [Section 3.5.5](#). If $\sigma^{4/3}$ does not exist, there also does not

exist a solution with makespan at most T , as shown in [Section 3.5](#). We will compute a solution σ with these properties, while minimizing the total load of setup times over all such solutions, and thus the total load of setup times and job pieces to be distributed on the m machines. This allows us to conclude that, if we are not able to compute σ , solution $\sigma^{4/3}$ does not exist for I , implying that there exists no solution with makespan T for I at all.

We begin the construction of σ by scheduling pairs of classes $i_8 \in \textcircled{8}$ and $i_9 \in \textcircled{9}$ on a single machine starting at $1/3$, respectively, until either $|\textcircled{8}| = 0$ or $|\textcircled{9}| = 0$, in time $\mathcal{O}(c)$. If there are not enough machines to do this, we can already conclude that there exists no schedule with makespan T . This also tells us which case we are in, i. e. if $|\textcircled{9}| = 0$ after this process, $\sigma^{4/3}$ must have the properties as described in [Section 3.5.4](#); if instead $|\textcircled{8}| = 0$ after this process, $\sigma^{4/3}$ must have the properties as described in [Section 3.5.5](#).

3.6.1 Classes of Type Eight Remain

Then $|\textcircled{9}| = 0$. We continue by constructing a bipartite graph, with a vertex for each class $i_8 \in \textcircled{8}$ and each class $i_{10} \in \textcircled{10}$, and an edge between two vertices iff one vertex represents a class of $\textcircled{8}$, the other a class of $\textcircled{10}$, and the combined load both classes is lesser or equal than $4/3$. We construct this graph and compute a maximal matching for it in time at most $\mathcal{O}(c^2)$ [52]. We schedule all classes of $i_8 \in \textcircled{8}$ and $i_{10} \in \textcircled{10}$ corresponding to the vertices of an edge in the maximal matching together on a single machine.

All pairs $i_8, i'_8 \in \textcircled{8}$ that are not yet scheduled, we schedule together on a single machine starting at $1/3$, leaving at most one class $i_8^* \in \textcircled{8}$ unscheduled. All remaining classes $i_{10} \in \textcircled{10}$ are scheduled alone a machine, each starting at time $1/3$. We call the set of machines with pairs of $\textcircled{8}$ classes and single $\textcircled{10}$ classes scheduled in σ on them $M_{8,10}$. If this is not possible, $\sigma^{4/3}$ does not exist. This leaves no class in $\textcircled{10}$, and at most the class $i_8^* \in \textcircled{8}$, unscheduled. The remaining machines with no setup times or job pieces scheduled on them yet we call M_{rem} .

Let $S_{\textcircled{7}}$ be the set of classes $i \in \textcircled{7}$ of I . We know that, if $\sigma^{4/3}$ exists, there exists a split of $S_{\textcircled{7}}$ into two sets S_1, S_2 of (partial) classes, with a setup time s_i in each set where i is (partly) included, such that all setup times and job pieces of S_1 are scheduled in intervals $[0, 1/3]$ of machines in $M_{8,10}$, and all setup times and job pieces of S_2 are either part of a nice instance, or scheduled in pairs on machines.

In fact, $s_i \leq 1/6$ holds for all setup times of classes i that are part of S_1 , and S_1 can be further divided into sub-sets $S_1^{>1/9}$, containing all setup times and job pieces of classes i with $s_i > 1/9$, and $S_1^{\leq 1/9}$ containing $S_1 \setminus S_1^{>1/9}$. According to $\sigma^{4/3}$, there exists some $0 \leq r' \leq |M_{8,10}|$, such that some classes $i \in S_1^{>1/9}$ with $1/6 < s_i + P(C_i) \leq 1/3$ are fully scheduled on a number of machines $r'_1 \leq r'$, some classes $i \in S_1^{\leq 1/9}$

with $2/9 < s_i + P(C_i) \leq 1/3$ are fully scheduled on a number of machines $r'_2 \leq |M_{8,10}| - r'$, and it holds for the sum of the load L_1 of all partial classes $i \in S_1^{>1/9}$, and the load L_2 of all remaining partial classes $i' \in S_1^{\leq 1/9}$ that

$$L_1 + L_2 \leq 1/6(r' - r_1) + 2/9(|M_{8,10}| - r_2 - r') .$$

S_2 can be further divided into two sets S'_2, S''_2 such that S'_2 is part of a nice sub-instance in $\sigma^{4/3}$, and S''_2 contains a number of classes i with $1/2 < s_i + P(C_i) \leq 5/9$ where an even number of these classes is scheduled in pairs on machines in M_{rem} , and one such class is scheduled together with i_g^* on a machines in M_{rem} if S''_2 is odd.

We know aim to compute a split of $S_{\text{⑦}}$ into two sets R_1, R_2 akin to S_1, S_2 , such that the load of R_1 , i. e. the load placed on machines $M_{8,10}$, is maximized over all such splits, while ensuring that the total load $L(R_1 + R_2)$ to be distributed stays small enough such that R_2 can still be feasibly placed on machines in M_{rem} . As the total load to be distributed depends only on the number of setup times used in a solution, this essentially boils down to minimizing the sum of all setup times in $R_1 \cup R_2$.

For each class $i \in [c]$, we compute the “obligatory” minimum load of each job j that must be scheduled as part of S_2 : For each j of class i , at most $1/3 - s_i$ processing time can be scheduled on machines in $M_{8,10}$ by the properties of $\sigma^{4/3}$. For each job j , this can be expressed as $p'_j = \max(p_j - 1/3 + s_i, 0)$, and as $P'(C_i) = P(C_i) - \sum_{j \in C_i} p'_j$ for each class i . We compute three ordered partial copies of $S_{\text{⑦}}$ in time $\mathcal{O}(c \log(c))$: One where all classes i with $s_i \leq 1/6$ and $P'(C_i) > 0$ are ordered non-decreasingly by $s_i/P'(C_i)$, one where all classes i with $1/6 < s_i + P(C_i) \leq 1/3$ are ordered non-increasingly by $s_i + P(C_i)$, and one where all classes i with $s_i \leq 1/9$ as well as $2/9 < s_i + P(C_i) \leq 1/3$ are ordered non-increasingly by $s_i + P(C_i)$.

We guess r' in at most c guesses. We set $r_1 = r_2 = 0$. While there still exists a class i with $s_i > 1/9$ and $1/6 < s_i + P(C_i) \leq 1/3$, and $r_1 < r'$, we remove the largest such i from $S_{\text{⑦}}$, schedule it in the interval $[0, 1/3]$ on some machine $q \in M_{8,10}$, and increase r_1 by 1. Similarly, while there still exists a class i with $s_i \leq 1/9$ and $2/9 < s_i + P(C_i) \leq 1/3$, and $r_2 < |M_{8,10}| - r'$, we remove the largest such i from $S_{\text{⑦}}$, schedule in the interval $[0, 1/3]$ on some machine $q \in M_{8,10}$, and increase r_2 by 1. Afterwards, we set $\omega = |M_{8,10}| - r_2 - r_1$, i. e. the number of machines in $M_{8,10}$ where the interval $[0, 1/3]$ is still empty, and set $r' = r' - r_1$. Let $F_1 := 1/6r'$ and $F_2 := 2/9(|\omega| - r')$.

Now, consider [Algorithm 7](#). We execute [Algorithm 7](#) first for the parameters $(Q_2, F_2, 0, 1/9)$, then for $(Q_1, F_1, 1/9, 1/6)$, and, if afterwards $L(Q_1) < F_1$, again for $(Q_1, F_1 - L(Q_1), 0, 1/9)$, in time $\mathcal{O}(n)$ each. Q_2 now contains at most all non-obligatory load of classes i with $s_i \leq 1/9$, Q_1 at most all non-obligatory load of classes i with $s_i \leq 1/6$. Let s_{e_1} and s_{e_2} be the setup times of the last classes C_{e_1} and C_{e_2} added to

Algorithm 7 (Q_k, F, ℓ, u)

```

1: while  $\exists i : \ell < s_i \leq u \wedge P'(C_i) = 0$  and  $F > 0$  do
2:   Add  $s_i$  to  $Q_k$ .
3:   if  $P(C_i) \leq F$  then
4:     Add  $C_i$  to  $Q_k$ .
5:     Set  $F = F - s_i - P(C_i)$ .
6:   else
7:     Add a set of job pieces  $J' \in C_i$  to  $Q_k$ , with  $P(J') = F$ .
8:     Set  $F = 0$ .
9:   end if
10: end while
11: while  $\exists i : \ell < s_i \leq u \wedge P'(C_i) > 0$  and  $F > 0$  do
12:   Let  $i$  be the class with smallest  $s_i / (P(C_i) - P'(C_i))$  over all such
   classes.
13:   Add  $s_i$  to  $Q_k$ .
14:   if  $P(C_i) - P'(C_i) \leq F$  then
15:     For each job  $j \in C_i$ , add a job piece  $j'$  with  $p_{j'} = p'_j$  to  $Q_k$ .
16:     Set  $F = F - s_i - (P(C_i) - P'(C_i))$ .
17:   else
18:     For each job  $j \in C_i$ , add a job piece  $j'$  with  $p_{j'} \leq p'_j$  to a set
      $J'$ , until  $P(J') = F$ .
19:     Add  $J'$  to  $Q_k$ .
20:     Set  $F = 0$ .
21:   end if
22: end while

```

Q_1 and Q_2 , respectively. We redefine Q_1, Q_2 as Wrap Sequences from their respective sets, starting with s_{e_1} and s_{e_2} , respectively, and with the rest of the (partial) classes in arbitrary order. We create two Wrap Templates ω^1, ω^2 , with $|\omega^1| = r'$, $|\omega^2| = |\omega| - r'$, and $\omega_r^k = (r, 0, 1/3)$ for $k \in \{0, 1\}$ and $1 \leq r \leq |\omega^i|$. We execute both WRAPLJ (Q_1, ω^1) and WRAPLJ (Q_2, ω^2) in time at most $\mathcal{O}(c)$. By definition of [Algorithm 7](#) and the fact that $s_{e_1} \leq 1/6$, we know that

$$\begin{aligned}
L(Q_1) &\leq 1/6|\omega^1| + s_{e_1} = 1/3|\omega^1| - 1/6|\omega^1| + s_{e_1} \\
&\leq 1/3|\omega^1| - 1/6(|\omega^1| - 1) \leq S(\omega^1) - \sum_{r=2}^{|\omega^1|} s_r .
\end{aligned}$$

The same is true for $L(Q_2)$, as $s_{e_2} \leq 1/9$ and thus

$$\begin{aligned}
L(Q_2) &\leq 2/9|\omega^2| + s_{e_2} = 1/3|\omega^2| - 1/9|\omega^2| + s_{e_2} \\
&\leq 1/3|\omega^2| - 1/9(|\omega^2| - 1) \leq S(\omega^2) - \sum_{r=2}^{|\omega^2|} s_r .
\end{aligned}$$

This implies that [Property 3](#) is fulfilled for both (Q_1, ω^1) and (Q_2, ω^2) . [Property 5](#) is fulfilled solely by definition of [Algorithm 7](#), as we only

add non-obligatory load of jobs, i. e. at most $1/3 - s_i$ processing time per job, to Q_1 and Q_2 respectively.

Let R_1 be the set of all setup times and job pieces of classes in $\mathcal{C}$ now placed on machines $M_{8,10}$, and R_2 the set of all job pieces not in R_1 , as well as one setup time per class of the job pieces in R_2 . We now aim to prove $L(R_2) \leq L(S_2)$, which will allow us to place R_2 feasibly on M_{rem} . In the following, we first assume that $L(Q_1) = F_1$ and $L(Q_2) = F_2$. If r' was guessed correctly, we now have $L(R_1) \geq L(S_1) + s_{e_1} + s_{e_2}$: We first place always the largest i with either $s_i > 1/9$ and $1/6 < s_i + P(C_i) \leq 1/3$ or $s_i \leq 1/9$ and $2/9 < s_i + P(C_i) \leq 1/3$ as long as there are machine available to place these classes on. By this, we ensure that both $r_1 \geq r'_1$ and $r_2 \geq r'_2$, and on both r'_1 and r'_2 machines in σ , there is at least as much load of $S_{\mathcal{C}}$ placed as on r'_1 and r'_2 machines in $\sigma^{4/3}$. On the $r' + (r_1 - r'_1)$ machines, which have at most $F_1 + (r_1 - r'_1)^{1/6}$ total load of $S_{\mathcal{C}}$ scheduled in $\sigma^{4/3}$, we place at least $F_1 + s_{e_1} + (r_1 - r'_1)^{1/6}$ load of $S_{\mathcal{C}}$: $F_1 + s_{e_1}$ by the execution of [Algorithm 7](#), and at least $(r_1 - r'_1)^{1/6}$ by placing i with $1/6 < s_i + P(C_i) \leq 1/3$ on machines $(r_1 - r'_1)$. Similarly, on $|M_{8,10}| - (r' + (r_2 - r'_2))$ machines, which have at most $F_2 + (r_2 - r'_2)^{2/9}$ load of $S_{\mathcal{C}}$ scheduled in $\sigma^{4/3}$, we place at least $F_2 + s_{e_2} + (r_2 - r'_2)^{2/9}$ load of $S_{\mathcal{C}}$: $F_2 + s_{e_2}$ by the execution of [Algorithm 7](#), and at least $(r_2 - r'_2)^{2/9}$ by placing i with $2/9 < s_i + P(C_i) \leq 1/3$ on machines $(r_2 - r'_2)$.

With this in hand, we can show that $L(R_2) \leq L(S_2)$: If there are no parts of classes i with $P'(C_i) > 0$ in R_1 , then the only setup times to appear in $R_1 \cup R_2$ more than once are s_{e_1} and s_{e_2} . Then $L(R_1) \geq L(S_1) + s_{e_1} + s_{e_2}$ directly implies $L(R_2) \leq L(S_2)$. Otherwise, we first add all possible classes with $P'(C_i) = 0$ to R_1 , and afterwards, minimize the total sum of setup times appearing both in R_1 and R_2 by always adding all non-obligatory load of the class i with current smallest $s_i/(P(C_i) - P'(C_i))$ to R_1 , until $L(R_1) = F$. Moreover, for all $0 \leq L \leq L(R_1)$, there exists some subset $R'_1 \subseteq R_1$ with $L(R'_1) = L$ that has the minimum total sum of setup times over all subsets with load L and the properties of S_1 : We get R'_1 by removing the setup times and job pieces with a total of $L(R_1) - L$ load that were last added to R_1 by [Algorithm 7](#), and thus obtain some $R'_1 \subseteq R_1$ with $L(R'_1) = L(S_1)$ which has the minimum total sum of setup times over all subsets of the $\mathcal{C}$ classes with load L . Thus, $L(S_1 \cup S_2) \geq L(R'_1 \cup R_2)$. As all setup times in job pieces in $R_1 \setminus R'_1$ can only reduce the load of R_2 , and $L(R_1) = F_1 + F_2 + s_{e_1} + s_{e_2} \geq L(S_1)$, $L(R_2) \leq L(S_2)$ follows.

If instead both $L(Q_1) < F_1$ and $L(Q_2) < F_2$, all non-obligatory load of classes with $s_i \leq 1/6$ and $s_i \leq 1/9$ is contained in R_1 , and, as $S_1^{>1/9}$ and $S_1^{\leq 1/9}$ contain at most this load, respectively, both $S_1^{>1/9} \subseteq R_1$ and $S_1^{\leq 1/9} \subseteq R_1$, immediately implying $L(R_2) \leq L(S_2)$. If either $L(Q_1) < F_1$ or $L(Q_2) < F_2$, the proof of $L(R_2) \leq L(S_2)$ is a special case of the

previous steps for $L(Q_1) = F_1$ and $L(Q_2) = F_2$, with $s_{e_1} = 0$ or $s_{e_2} = 0$ and considering the fact that $S_1^{>1/9} \subseteq R_1$ or $S_1^{\leq 1/9} \subseteq R_1$, respectively.

Now, let $R_2'' \subseteq R_2$ be the subset of classes i that are fully included in R_2 with $5/9 \geq s_i + P(C_i) > 1/2$, and $R_2' = R_2 \setminus R_2''$. We schedule an even number of classes $i \in R_2''$ in pairs on machines of M_{rem} . If a single class $i \in R_2''$ remains afterwards, we schedule it together with i_8^* on a single machine of M_{rem} , if i_8^* exists and $L(i_8^*) + L(i) > 1$, and otherwise add both i and i_8^* to R_2' . We then create a nice instance $I_{\text{nice}} = (S_{\text{nice}}, M_{\text{nice}})$, with S_{nice} consisting of all classes of types $(\textcircled{1}, \dots, \textcircled{6})$, i_8^* if it exists and is not yet scheduled, and R_2' . M_{nice} consists of all machines of M_{rem} with no setup times or jobs scheduled on them yet.

If $|M_{\text{nice}}| > m_{\text{nice}}(I_{\text{nice}})$, we can conclude $\sigma^{4/3}$ does not exist: In $\sigma^{4/3}$, all classes of types $\textcircled{1}, \dots, \textcircled{6}$ are all part of a nice sub-instance $I'_{\text{nice}} = (S'_{\text{nice}}, M'_{\text{nice}})$ with $|M'_{\text{nice}}| = |M_{\text{rem}}| - \lceil |S_2''|/2 \rceil$. S'_{nice} also includes all setup times and job pieces of S_2' , but not S_2'' , and i_8^* only if S_2'' is even. Let $m' = \lceil |S_2''|/2 \rceil$. Then, for each of the m' machines of M_{rem} that have a pair of classes in S_2'' scheduled on it in $\sigma^{4/3}$, there is a $1/9$ free space on machines in $M_{8,10}$ in $\sigma^{4/3}$. In $\sigma^{4/3}$, each time two classes in S_2'' are combined, a machine is added to $M_{8,10}$, but only setup times and job pieces with load $1/9$ are added to S_1 , allowing for a placement of $2/9$ load on this machine. Thus, $L(R_1) \geq L(S_1) + m'1/9$, and $L(R_2) - m'1/9 \leq L(S_2)$, which in turn implies $L_{\text{nice}}(I_{\text{nice}}) - m'1/9 \leq L_{\text{nice}}(I'_{\text{nice}})$. As there is at most $10/9$ load on each of the m' machines, the total load on machines M_{rem} in $\sigma^{4/3}$ is $L_{\text{nice}}(I'_{\text{nice}}) + 10/9m' \geq L_{\text{nice}}(I_{\text{nice}}) - m'1/9 + 10/9m' = L_{\text{nice}}(I_{\text{nice}}) + m'$. As we schedule at least 1 load on all machines in $M_{\text{rem}} \setminus M_{\text{nice}}$, in total, this implies $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$, if $\sigma^{4/3}$ exists. Therefore, in any case, [Theorem 12, \(ii\)](#) is fulfilled, and we can compute a $4/3$ -approximation for the nice sub-instance in time $\mathcal{O}(n)$. As all other machines have load at most $4/3$, and we have now scheduled all jobs in the instance, the resulting schedule σ is a $4/3$ -approximation of the whole instance. The run time is dominated by the matching sub-routines, which take time at most $\mathcal{O}(c^2) \in \mathcal{O}(n^2)$ each, resulting in a run-time of at most $\mathcal{O}(n^2)$ for the whole algorithm for some fixed T , and a total run time of $\mathcal{O}(n^2 \log(1/\varepsilon))$.

3.6.2 Classes of Type Nine Remain

Then $|\textcircled{8}| = 0$. We know that in $\sigma^{4/3}$, each class $i_{9,10} \in \textcircled{9} \cup \textcircled{10}$ is scheduled on a single machine, together with some jobs j of classes of $\textcircled{7}$, and the rest of the machines are part of a nice instance $I'_{\text{nice}} = (S'_{\text{nice}}, M'_{\text{nice}})$, if it exists, where M'_{nice} consists of all machines without machines of $\textcircled{8}, \textcircled{9}$ or $\textcircled{10}$ scheduled on them. We thus create a set $M_{9,10}$ of $|\textcircled{9} \cup \textcircled{10}|$ yet unused machines, and a set M_{nice} from all other unused machines. Then $|M_{\text{nice}}| = |M'_{\text{nice}}|$ by construction of $\sigma^{4/3}$. We initially schedule all classes $i_{9,10} \in \textcircled{9} \cup \textcircled{10}$ on a single machine of $M_{9,10}$, respectively, each class starting at 0. We now want to build a nice

instance $I_{\text{nice}} = (S_{\text{nice}}, M_{\text{nice}})$ for I , such that $L_{\text{nice}}(S_{\text{nice}}) \leq L_{\text{nice}}(S'_{\text{nice}})$; this immediately implies the computability of a $4/3$ -approximation for I_{nice} by the computability of such a solution for I'_{nice} for $\sigma^{4/3}$, if it exists. Let $S_{\textcircled{7}}$ be the set of classes $i \in \textcircled{7}$ of I .

We also know that, for each $i \in \textcircled{7}$ with a job $j \in C_i : 2s_i + p_j > 2/3$, that $s_i \in S'_{\text{nice}}$. We thus also add s_i to S_{nice} for each such i . In $\sigma^{4/3}$, there is some number x such that only the x machines q of $M_{9,10}$ with largest $F_q = 1 - L(i_{9,10})$ have jobs j with $2s_j + p_j > 2/3$ scheduled on them, fully or partly, together with the classes $i_{9,10}$. We add all jobs j with $2s_j + p_j > 2/3$ to a set J , order J by non-decreasing s_j , order all $q \in M_{9,10}$ non-increasingly by F_q , each in time $\mathcal{O}(n \log(n))$, and finally guess this number x (at most n guesses). Let $M'_{9,10}$ be the set of the first x machines of $M_{9,10}$. While for the first $j \in J$ and for the first two machines $q, q' \in M'_{9,10}$, it holds that $F_q + F_{q'} \geq s_j + 1/3$, we (re-)schedule $i_{9,10}$ on q such that it finishes at exactly $4/3$, $i'_{9,10}$ on q' such that it starts at 0, s_j as well as at most $4/3 - L(i_{9,10}) - s_j$ processing time of j on q starting at 0, and s_j as well as $p_j - (4/3 - L(i_{9,10}) - s_j)$ processing time of j on q' , starting at $L(i'_{9,10})$. If, after this process, $M'_{9,10} \neq \emptyset$, we schedule a setup s_j , as well as $1/3 - s_j$ processing time of the first $j \in J$ on the first machine $q \in M'_{9,10}$, starting at 0, and $i_{9,10}$ starting at $1/3$. We remove all such placed job pieces from $S_{\textcircled{7}}$.

We set $M_{9,10} = M_{9,10} \setminus M'_{9,10}$, and proceed similarly to $\sigma^{4/3}$: On all machines $1, \dots, |M_{9,10}| \in M_{9,10}$, we (re-)schedule all remaining classes $i_{9,10} \in \textcircled{9} \cup \textcircled{10}$ alternately starting at 0 or finishing at $4/3$, such that on machine $|M_{9,10}|$, $i_{9,10}$ finishes at $4/3$. This schedules all remaining classes in $\textcircled{9} \cup \textcircled{10}$ by definition of $M_{9,10}$. We define a Wrap Template $\omega = (\omega_1, \dots, \omega_{|M_{9,10}|})$, with $a_r = 0$ and $b_r = 4/3 - L(i_{9,10})$ if $i_{9,10}$ finishes at $4/3$ on machine r , or $a_r = 4/3 - L(i_{9,10})$ and $b_r = 4/3$ if $i_{9,10}$ starts at 0 on machine r . Let $F = \sum_{r=1}^{|\omega|} (b_r - a_r - 1/3)$ be the total space on machines $1, \dots, |\omega|$ available in σ besides the classes of $\textcircled{9} \cup \textcircled{10}$ and the $1/3$ space giving by the approximation error. Let Q be an empty Wrap Sequence. We first execute [Algorithm 8](#) to fill Q .

Algorithm 8

```

1: while  $\exists i : s_i \notin Q \wedge s_i + P(C_i) \leq 1/3$  and  $F > 0$  do
2:   Algorithm 9 ( $i$ ).
3: end while
4: while  $\exists i : s_i \notin S_{\text{nice}} \wedge s_i \notin Q \wedge s_i + P(C_i) > 1/3$  and  $F > 0$  do
5:   Let  $i$  be the class with smallest  $s_i$  over all such classes.
6:   Algorithm 9 ( $i$ ).
7: end while
8: while  $\exists i : s_i \in S_{\text{nice}} \wedge s_i \notin Q \wedge s_i + P(C_i) > 1/3$  and  $F > 0$  do
9:   Let  $i$  be the class with smallest  $s_i / (P(C_i) - \sum_{j \in J} p_j)$  over all such
   classes.
10:  Algorithm 9 ( $i$ ).
11: end while

```

Algorithm 9 (i)

```

1: Put  $s_i$  into  $Q$  and set  $F := F - s_i$ .
2: while  $F > 0$  and  $\exists j \in S_{\textcircled{7}} \cap C_i$  do
3:   Remove  $j$  from  $S_{\textcircled{7}}$ .
4:   if  $F > p_j$  then
5:     Add  $j$  to  $Q$  and set  $F = F - p_j$ .
6:   else
7:     Split  $p_j$  into two parts  $j^1$  and  $j^2$  with  $p_{j^1} = F$  and  $p_{j^2} = p_j - F$ .
8:     Add  $j^1$  to  $Q$ ,  $j^2$  to  $S_{\text{nice}}$ , and set  $F := 0$ .
9:   end if
10: end while

```

We then execute $\text{WRAPLJ}(Q, \omega)$. [Property 1](#) is fulfilled: $L(Q) \leq S(\omega) - \sum_{r=2}^{|\omega|} s_r$ is true as $\sum_{r=2}^{|\omega|} s_r \leq (|\omega| - 1)^{1/3}$ by definition of classes in $S_{\textcircled{7}}$, we know that $L(Q) \leq F$ by definition of [Algorithm 8](#) and [Algorithm 9](#), and thus $S(\omega) = F + (|\omega| - 1)^{1/3}$. The statement $b_r - a_r \geq 1/3 \geq \max_{r \in |\omega|} s_r$ also holds by definition of classes in $S_{\textcircled{7}}$. The schedule produced by $\text{WRAPLJ}(Q, \omega)$ is also never parallelized, as only jobs j of classes i with $2s_i + p_j \leq 2/3$ are placed, and, by our definition of ω , no matter where a job begins, the next $2/3$ space in ω does not overlap. We add all remaining jobs of $S_{\textcircled{7}}$ to S_{nice} , and add a setup time for each class that has job pieces but no setup time in S_{nice} , to S_{nice} .

Again, there is at most one job j of some class e that is only partly included in Q , where e is the class that was considered last in [Algorithm 8](#), and there is at least $1/3$ free space on machine ω_1 by definition of ω . Let p'_j be the processing time of all job pieces of j in Q , p''_j the processing time of all job pieces of j in S_{nice} . If $s_e + P(C_e) \leq 1/3$, we schedule e fully on ω_1 . Else, we slightly modify ω by setting $a_1 = 0$, i. e. increasing the total space of ω by $1/3$. As $2s_i + p_j \leq 2/3$, either $p'_j \leq 1/3$ or $p''_j \leq 1/3$ must hold. If $p''_j \leq 1/3$, we can then simply add j fully to Q without violating any of the Batch Wrapping properties. If instead $p'_j \leq 1/3$, we modify ω again by setting $b_{|\omega|} = 1/3$. This still guarantees that $S(\omega) \geq F$, but now also makes sure that all of p'_j is scheduled in the interval $[0, 1/3]$ on machine $|\omega|$; as [Theorem 12](#) places all jobs of $\textcircled{7}$ classes above $1/3$, this guarantees that no parallelization occurs.

Now, it is only left to show that if $\sigma^{4/3}$ exists, indeed $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$ holds. As all machines of $M_{9,10}$ have load at least 1 in our solution, $L_{\text{nice}}(I_{\text{nice}}) > |M_{\text{nice}}|$ can only occur if the total load of setup times in our solution is greater than in $\sigma^{4/3}$ and $L(Q) = F$. Recall that J is the set of all jobs j of classes $i \in \textcircled{7}$ with $2s_i + p_j > 2/3$. When filling machines $M'_{9,10}$, there is exactly one setup time of some job $j \in J$ per machine of $M'_{9,10}$ in $\sigma^{4/3}$. As we always schedule jobs of J

with smallest overall setup time first, we minimize the total load of setup times when scheduling jobs of J .

Let Q' be the Wrap Sequence of $\sigma^{4/3}$ for Batch-Wrapping on machines $M_{9,10} \setminus M'_{9,10}$, and $I'_{\text{nice}} = (S'_{\text{nice}}, M_{\text{nice}})$ the nice instance of $\sigma^{4/3}$. When adding classes to Q to be Batch-Wrapped on machines $M_{9,10} \setminus M'_{9,10}$, we first add all classes i with $s_i + P(C_i) \leq 1/3$, then all classes i' which have no job $j \in J$ (exactly the classes i' with $s_{i'} \notin S_{\text{nice}}$) in non-decreasing order of their setup, and finally, classes i'' that do have at least one job $j \in J$ in non-decreasing order of $s_i/(P(C_i) - \sum_{j \in J} p_j)$.

If Q contains only classes i , then there is exactly one setup time per class in $Q \cup S_{\text{nice}}$, due to our placement of classes e with $s_e + P(C_e) \leq 1/3$, which is the minimum number of setup times over all such solutions. If Q also contains classes i' but not i'' , then $e = i'$ for some i' . If s_e both in Q' and S'_{nice} , we are done. If s_e in S'_{nice} , but not in Q' , then either $L(Q) - s_e \geq L(Q')$, as in this case, Q' contains only full classes and does not use the additional $1/3$ space on machine ω_1 , or there is some $s_{e'}$ with $s_{e'} \in Q'$ and $s_{e'} \geq s_e$ by our ordering of i' . In the former case, the total load of setup times in our solution is increased by s_e compared to $\sigma^{4/3}$, but we schedule s_e more load in Q than in Q' , and thus, $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$ if $\sigma^{4/3}$ exists. In the latter case, the total load of setup times in our solution is smaller or equal than in $\sigma^{4/3}$, and $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$ as well. If Q contains also classes i'' , then $e = i''$ for some i'' , and again, either $L(Q) - s_e \geq L(Q')$, or Q' contains classes i'' as well. By our ordering of i'' and prior scheduling of all classes i and i' , the total load of setup times of classes i'' is minimized over all such solutions, and $L_{\text{nice}}(I_{\text{nice}}) \leq |M_{\text{nice}}|$ if $\sigma^{4/3}$ exists.

Thus, in any case, [Theorem 12](#), (ii) is fulfilled, and we can compute a $4/3$ -approximation for the nice sub-instance in time $\mathcal{O}(n)$. As all other machines have load at most $4/3$, and we have now scheduled all jobs in the instance, the resulting schedule σ is a $4/3$ -approximation of the whole instance. All orderings can be computed before guessing the size x of $M'_{9,10}$ in time $\mathcal{O}(n \log(n))$. The number of guesses on x is at most n ; for each guess, all steps can be computed in total time $\mathcal{O}(n)$, resulting in a run time of at most $\mathcal{O}(n^2)$ for the whole algorithm for some fixed T , and a total run time of $\mathcal{O}(n^2 \log(1/\varepsilon))$.

3.7 CONCLUSION

In this chapter, we have managed to give a $(4/3 + \varepsilon)$ -approximate algorithm for $P|\text{pmnt, setup} = s_i|C_{\max}$ with run time in $\mathcal{O}(n^2 \log(1/\varepsilon))$. For any $\varepsilon < 1/6$, this improves upon the previously best known approximation factor of $3/2$ of a polynomial time algorithm by Deppert and Jansen [\[41\]](#) for this problem.

Deppert and Jansen [\[41\]](#) manage to get rid of the ε -error on the solution by introducing an idea for the binary search called “Class Jumping”, by which they manage to bound the number of guesses

needed by the binary search to find the actual optimal makespan OPT in terms of $\mathcal{O}(\log(cm)) \in \mathcal{O}(\log(n))$. Employing this or a similar idea to our approach, one could improve the approximation factor to $4/3$, with the run time likely being in $\mathcal{O}(n^2 \log(n))$. “Class Jumping” is however based on finding a makespan value T acting as a lower bound on the optimum, $T \leq \text{OPT}$, for which Deppert and Jansen [41] are then able to compute a solution with makespan at most $3/2T$. Our approach however is based on structural properties of optimal solutions, i. e. we can only compute solutions for values $T \geq \text{OPT}$. To use this idea, we would thus need to find not a lower bound, but the exact value $T = \text{OPT}$, which is significantly harder. If this can be done in time $\text{poly}(|I|)$ is an open question.

Also, while there do exist some EPTASs for the splittable, non-preemptive, and preemptive with single-job-batches variants of the considered problem [77], the running times of these algorithms all have a high exponential dependency on $1/\varepsilon$, as well as a large polynomial dependence on $|I|$, e. g. the running time of the EPTAS for the non-preemptive variant is $2^{\mathcal{O}(\varepsilon^{-3} \log^4(\varepsilon^{-1}))} |I|^4 \log(|I|)$ in the worst case. Subsequently, Deppert and Jansen [41] gave $3/2$ -approximate algorithms for all these problem variants, with running time in $\mathcal{O}(n \log(n))$, which, in general, all have worse approximation ratios than the EPTASs, but may be more usable in practice thanks to the low computational complexity. As the approaches by Deppert and Jansen [41] are quite similar for all problem variants, we pose the question if our approach as well can be modified to obtain $(4/3 + \varepsilon)$ -approximate algorithms for the splittable and non-preemptive variant as well, with practically usable running time in $\mathcal{O}(n^2 \log(1/\varepsilon))$.

Of even greater interest would be to find a PTAS for the here studied problem $\text{P|pmnt, setup} = s_i |C_{\max}$, which is known for all the important variations of this problem, but not $\text{P|pmnt, setup} = s_i |C_{\max}$ itself. We hope our here presented structural analysis of this problems provides approaches for future research in this direction.

Part II

PARAMETERIZED ALGORITHMS AND APPROXIMATION SCHEMES

FPT ALGORITHMS FOR ENVY MINIMIZATION ON UNIFORM MACHINES

4.1 INTRODUCTION

In this chapter, we consider scheduling on uniform machines $HM(m, n)$ encoding, with the objective of minimizing C_{envy} , which, by its definition, is highly related to both objectives C_{max} and C_{min} . Formal definitions of $HM(m, n)$ encoding, as well as the three objective functions C_{max} , C_{min} and C_{envy} can be found in [Section 1.1](#), “Scheduling on Uniform Machines”. The objective C_{envy} was first introduced by Lipton et al. [98], and is meant to measure some kind of fairness of an allocation.

Let p_{max} as usual be the largest occurring job size in an instance, i. e. $p_{\text{max}} = \max_{j \in [d]} p_j$. Here, we are interested in the structure of optimal solutions if p_{max} is small and there are only a few job types d . This occurs e. g. in approximative settings after rounding or when scheduling maintenance for air crafts [103]. The problem is $\mathcal{NP}$ -hard for all three mentioned objective functions [123] and hence unlikely to admit a polynomial time algorithm (unless $\mathcal{P} = \mathcal{NP}$). However, there might be (and in fact, there are) polynomial time algorithms if d and p_{max} are fixed, i. e. algorithms that are either XP, or even FPT, w. r. t. parameters d and p_{max} .

Recall that, in contrast to low-multiplicity scheduling, where $\text{poly}(n)$ is polynomial in the encoding of the input, for high-multiplicity encoding, $\text{poly}(n)$ is exponential in the encoding of the input. So, when designing FPT algorithms for instances I in $HM(m, n)$ encoding, we have $\text{poly}(|I|) = \text{poly}(\log n, \log m, \tau, d)$. Also recall that any low-multiplicity encoded instance I can be transformed into a $HM(n)$ encoded one, and any $HM(n)$ encoded instance I can be transformed into a $HM(m, n)$ encoded one, in $\mathcal{O}(\text{poly}(|I|))$ time each. This implies that any algorithm that computes solutions for $HM(m, n)$ encoded instances in time $f(|I|) \cdot \mathcal{O}(\text{poly}(|I|))$ can be used to compute solutions for $HM(n)$ encoded instances in time $f(|I|) \cdot \mathcal{O}(\text{poly}(|I|))$, and any algorithm that computes solutions for $HM(n)$ encoded instances in time $f(|I|) \cdot \mathcal{O}(\text{poly}(|I|))$ can be used to compute solutions for low-multiplicity encoded instances in time $f(|I|) \cdot \mathcal{O}(\text{poly}(|I|))$ as well, for any computable function f .

4.1.1 Related Work

Early works regarding high-multiplicity scheduling usually considered a single machine setting, e. g. [59, 63, 66, 67, 68, 108, 109]. This is inter alia due to the fact that for multiple machines and $HM(m, n)$ encoding, one cannot output the jobs assigned to a machine for each machine individually, as this would take $\Omega(m)$ space and hence $\Omega(m)$ time. So, in order to handle multiple machines, a compact output encoding has to be used, usually together with some statements that show that there is always a solution that can be encoded in such a compact way. An approach of this kind was used by McCormick et al. [103]. Clifford and Posner [32] showed in 2001 that for objective $C_{\max}$ on uniform machines there is always a solution where only $\min\{m, \tau 2^d\}$ different job assignments occur, which allows a sufficiently compact encoding for FPT algorithms with any parameter $k \geq d$. This argument can easily be extended to the objectives $C_{\min}$ and C_{envy} . Interestingly, the same argument was independently rediscovered in 2006 by Eisenbrand and Shmonin [47]. They used it to show that the so called “one dimensional Cutting Stock Problem”, a variation of bin packing with a high-multiplicity encoding of the items, is in $\mathcal{NP}$; this has been a long standing open question. Goemans and Rothvoß [56] proved that the one dimensional cutting stock problem can be solved in polynomial time for a fixed number d of different item sizes, that is, the problem with parameter d lies in XP . They also applied this result to $HM(m, n)$ scheduling; in particular, they showed that with a compact encoding, makespan minimization on uniform machines with parameter $d + \tau$ lies in XP . Koutecký and Zink [94] showed that even when only the jobs are encoded in a compact way, makespan minimization on uniform machines is already $\mathcal{NP}$ -hard for 6 job types. Hence, there is no FPT algorithm with parameter (only) d for this problem, unless $\mathcal{P} = \mathcal{NP}$.

Knop et al. [92] proposed a technique for solving N -fold ILPs where the blocks are encoded in a high-multiplicity way. Their approach is based on proximity of those ILPs, using quite involved techniques. Applied to $HM(m, n)$ scheduling by encoding the problem as a [Configuration-ILP](#), which does have N -fold structure, and choosing appropriate configuration sets for all $t \in [\tau]$ and the respective objective function, one achieves a run time of $p_{\max}^{\mathcal{O}(d^2)} \cdot \text{poly}(\log n, \log m, \tau, d)$ for $\text{Q}|HM(m, n)|C_{\max}$ and $\text{Q}|HM(m, n)|C_{\min}$. Knop and Koutecký [89] show that $HM(m, n)$ scheduling on unrelated machines (and thus also on uniform machines) admits a kernel polynomial in $p_{\max}$ and the number of different machine types τ , computable in time polynomial in these parameters.

Regarding $HM(n)$ encoding, there exists a series of FPT algorithms parameterized by $p_{\max}$ and d with improving run times for both these problems, which stem directly from improving run times of N -fold

ILP solvers [38, 46, 90]. Finally, problem-specific algorithms with run time $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$ were given by Rohwedder [111] for $Q|HM(n)|C_{\max}$, and by Jansen et al. [75] for $Q|HM(n)|C_{\min}$, respectively, which is asymptotically the best possible run time for these parameters under the ETH [31, 76].

4.1.2 Our Contribution

We develop a novel binary search routine for C_{envy} , where the naive approach did not lead to a (satisfying) result so far. Combining this routine with the result of Knop et al. [92] we derive a $p_{\max}^{\mathcal{O}(d^2)} \text{poly}(\log n, \log m, \tau, d)$ time algorithm for $Q|HM(m, n)|C_{\text{envy}}$. This, to the best of our knowledge, is the first result showing that $Q||C_{\text{envy}}$ is FPT only in the parameter $p_{\max}$, in the high-multiplicity setting or otherwise.

We further manage to improve the run time for solving the same problem $Q|HM(m, n)|C_{\text{envy}}$ to $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$, using techniques introduced by Rohwedder [111] as well as our previous results for $Q|HM(m, n)|C_{\text{envy}}$. This is the asymptotically best possible run time for these parameters under the ETH, matching the run times from the relevant literature for solving both $Q|HM(n)|C_{\max}$ [111] and $Q|HM(n)|C_{\min}$ [75].

ORGANIZATION OF THIS CHAPTER We start by giving some preliminary notions in Section 4.2, stating the basic notation used throughout this chapter and stating some results from the literature that we use for our work. In Section 4.3, we prove our first main result, the $p_{\max}^{\mathcal{O}(d^2)} \text{poly}(\log n, \log m, \tau, d)$ time algorithm for $Q|HM(m, n)|C_{\text{envy}}$. In Section 4.4, we follow up on this by introducing the speed-up for solving $Q|HM(m, n)|C_{\text{envy}}$ in time $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$. We finish by giving a short conclusion and an outlook on future research in Section 4.5.

4.2 PRELIMINARIES

NOTATION By convention, any lower case letter with a subscript index is never a vector, but the component of a vector. For two vectors v, w we write $v \leq w$ if for all i we have $v_i \leq w_i$. We use $\lfloor v \rfloor$ to denote the vector that one gets by replacing every component v_i of v by $\lfloor v_i \rfloor$. We use the same idea of point-wise rounding to define $\lfloor f \rfloor$ for a map f . Analogously, $\lceil \cdot \rceil$ is defined for vectors and maps. In the context of $\max(\cdot)$, $\sum(\cdot)$, et cetera we use the same notation for vectors that is used for sets, e. g. write $\sum v$ for $\sum_{i \in [d]} v_i$. We use $x \equiv_a y$ to denote $x \equiv y \pmod{a}$.

A common approach in scheduling is to guess the optimal objective value T via binary search, and then check if there exists a solution whose objective value is at least as good as T , in effect converting the optimization problem into a series of decision problems. This

approach works for a broad set of objectives and in particular for $C_{\min}$, $C_{\max}$, and C_{envy} . To then compute solutions with this objective value, often the previously defined [Configuration-ILP](#) is used, which is based on the concept of configurations, i. e. all ways to place jobs with total processing time at least some lower bound ℓ and at most some upper bound u (cf. [Section 1.1](#)). It encodes that for each $t \in [\tau]$, exactly μ_t configurations from the set $\mathcal{C}(\ell_t, u_t)$ have to be chosen, such that the total sum of assigned jobs over all chosen configurations and over all $t \in [\tau]$ is exactly v , where ℓ_t and u_t are problem and machine type specific lower and upper bounds, respectively. For that, a variable $x_{t,c}$ is used for each $t \in [\tau]$ and each $c \in \mathcal{C}(\ell_t, u_t)$, indicating that configuration c is chosen exactly $x_{t,c}$ times for machines of type t .

Setting e.g. $u_t = s_t T$ for all $t \in [\tau]$, a feasible solution to the [Configuration-ILP](#) then implies a schedule with $C_{\max} \leq T$, while infeasibility implies $C_{\max} > T$; the same holds for setting $\ell_t = s_t T$ for all $t \in [\tau]$ and $C_{\min}$. An optimal solution can then be computed via simple Binary Search on the possible values of T . A feasible solution to [Configuration-ILP](#) when setting $u_t = s_t T_1$ and $\ell_t = s_t T_2$ for all $t \in [\tau]$ implies a schedule with $C_{\text{envy}} \leq T_1 - T_2$. Note that for C_{envy} , the choice of T_1 and T_2 does not however directly imply $C_{\text{envy}} > T_1 - T_2$, as there may exist some T'_1, T'_2 such that [Configuration-ILP](#) is feasible with $u_t = s_t T'_1$ and $\ell_t = s_t T'_2$, and simultaneously $T'_1 - T'_2 \leq T_1 - T_2$. This prevents the direct application of a simple Binary Search for C_{envy} . [Configuration-ILP](#) has N -fold structure with block parameters $R = d$, $H = \max_{t \in [\tau]} |\mathcal{C}(\ell_t, u_t)|$, $S = 1$, $N = \tau$, and $\|A\|_\infty$ equaling the largest entry in any $c \in \mathcal{C}(\ell_t, u_t)$ over all $t \in [\tau]$. As we repeatedly use the following application of a result by Knop et al. [\[92\]](#) for solving [Configuration-ILP](#), we formally state it here.

Lemma 12 (Knop et al. [\[92\]](#), Application of Theorem 1). *For any ℓ_t, u_t with $\langle \ell_t, u_t \rangle = \text{poly}(\log n, \log m, \log p_{\max}, \tau, d)$ for all $t \in [\tau]$, [Configuration-ILP](#) can be solved in time $p_{\max}^{\mathcal{O}(d^2)} \cdot \text{poly}(\log n, \log m, \tau, d)$.*

Proof. The P -representation ([\[92\]](#), Definition 1) of [Configuration-ILP](#), representing the configurations for each $t \in [\tau]$ as integer points of a rational polytope, has parameters $M = 1$, $d = d$, $\Delta = p_{\max}$ and $L = \langle p_{\max}, (u_1 - \ell_1), \dots, (u_\tau - \ell_\tau) \rangle$. According to Knop et al. [\[92\]](#), Theorem 1.3, [Configuration-ILP](#) can then be solved in time $(dp_{\max})^{\mathcal{O}(d^2)} (L + \langle n, \sum_{i=1}^\tau \mu_i \rangle)^{\mathcal{O}(1)}$, which is in $p_{\max}^{\mathcal{O}(d^2)} \cdot \text{poly}(\log n, \log m, \tau, d)$ if $\langle \ell_t, u_t \rangle = \text{poly}(\log n, \log m, \log p_{\max}, \tau, d)$. $\square$

4.3 $Q|HM(m, n)|C_{\text{envy}}$ PARAMETERIZED BY $p_{\max}$

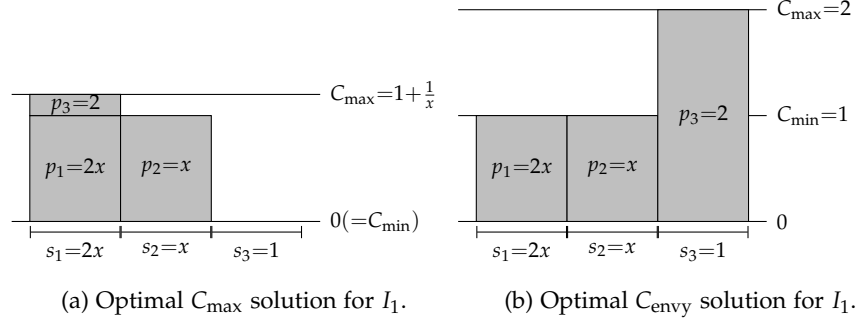
In this section, we present the $p_{\max}^{\mathcal{O}(d^2)} \text{poly}(\log n, \log m, \tau, d)$ time algorithm for solving $Q|HM(m, n)|C_{\text{envy}}$. The main difficulty for C_{envy} compared to $C_{\max}$ and $C_{\min}$ is that, given some k , it is not that simple to decide if $k \leq \text{OPT}$. This is mainly due to the fact that in order to

use a [Configuration-ILP](#), we do need lower and upper bounds for the completion times. But finding these is not that simple, as from infeasibility of an instance with lower bound $T - k$ and upper bound T for some guess T , it is not clear if we need to increase or decrease T , in contrast to C_{max} and C_{min} , where this is very clear. Even given an optimal C_{max} (resp. C_{min}) value for an instance, one can not simply fix this value as an upper (resp. lower) bound, and do a binary search for the C_{min} (resp. C_{max}) value to guess C_{envy} , as the C_{max} (resp. C_{min}) value of an optimal solution minimizing this value may differ from the C_{max} (resp. C_{min}) value of an optimal solution minimizing C_{envy} .

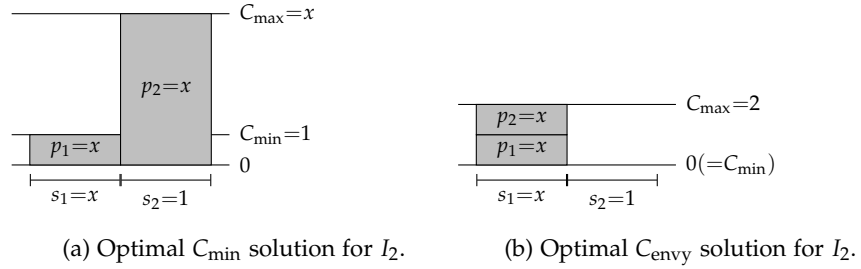
To give some intuition on this difficulty, we give two Scheduling instances I_1 and I_2 on Uniform Machines, where, for I_1 , the C_{max} -value of an optimal solution minimizing C_{max} differs from the C_{max} -value of an optimal solution minimizing C_{envy} , and for I_2 , the C_{min} -value of an optimal solution maximizing C_{min} differs from the C_{min} -value of an optimal solution minimizing C_{envy} . The general idea behind these instances is that, for minimizing C_{max} (resp. maximizing C_{min}), an optimal solution may neglect the C_{min} (resp. C_{max}) value to get a better objective value, whereas for C_{envy} , both C_{max} and C_{min} values are crucial for the objective value. For readability purposes, we describe both instances in Low-Multiplicity encoding, and present visualizations in [Figure 4.1](#) and [Figure 4.2](#), respectively.

Let I_1 consist of three jobs with processing times $p_1 = 2x, p_2 = x, p_3 = 2$, and three machines with speeds $s_1 = 2x, s_2 = x, s_3 = 1$, for some $x > 1$. An optimal solution minimizing C_{max} has $C_{\text{max}} = 1 + \frac{1}{x}$, by putting the jobs with respective processing times $2x$ and 2 on the machine with speed $2x$, and the job with processing time x on the machine with speed x (cf. [Figure 4.1a](#)). Since the third machine with speed 1 does not get any jobs in this schedule however (putting any job on this machine would result in $C_{\text{max}} \geq 2 > 1 + \frac{1}{x}$ for $x > 1$), $C_{\text{min}} = 0$, and thus, $C_{\text{envy}} = C_{\text{max}} - C_{\text{min}} = 1 + \frac{1}{x}$. An optimal solution minimizing C_{envy} has $C_{\text{envy}} = C_{\text{max}} - C_{\text{min}} = 2 - 1 = 1$, by putting the job with processing time $2x$ on the machine with speed $2x$, the job with processing time x on the machine with speed x , and the jobs with processing time 2 on the machine with speed 1 (cf. [Figure 4.1b](#)). This results in $C_{\text{max}} = 2 > 1 + \frac{1}{x}$, i.e. a greater makespan than the optimal solution for C_{max} .

Let I_2 consist of two jobs with processing times $p_1 = p_2 = x$, and two machines with speeds $s_1 = x$ and $s_2 = 1$, for some $x > 3$. An optimal solution maximizing C_{min} has $C_{\text{min}} = 1$, by putting one job with processing times x on the machine with speed x , and the other job with processing time x on the machine with speed 1 (cf. [Figure 4.2a](#)). This implies $C_{\text{max}} = x$, and thus $C_{\text{envy}} = C_{\text{max}} - C_{\text{min}} = x - 1 > 2$, for $x > 3$. An optimal solution minimizing C_{envy} has $C_{\text{envy}} = C_{\text{max}} - C_{\text{min}} = 2$, by putting both jobs with processing times x on the machine with speed x , and no jobs on the machine with speed

Figure 4.1: Instance I_1 with $x = 5$.

1 (cf. Figure 4.2b). This results in $C_{\min} = 0$, i.e. a lesser minimum completion time than the optimal solution for $C_{\min}$.

Figure 4.2: Instance I_2 with $x = 5$.

Many approaches to tackle this problem have run time $\Omega(s_{\max} + p_{\max})$, where $s_{\max}$ is the maximum speed of any machine occurring in an instance, which is not FPT with respect to (only) $p_{\max}$. We propose the first approach to solve this problem without the need of introducing an additional parameter¹, which eventually leads to the first FPT algorithm with parameter $p_{\max}$ for $Q||C_{\text{envy}}$. Remember that

$$C_{\max}(\sigma) := \max_{i \in [m]} C_i^\sigma \quad \text{and} \quad C_{\min}(\sigma) := \min_{i \in [m]} C_i^\sigma,$$

where $C_i^\sigma = s_i^{-1} p^\top \sigma_i$ is the completion time of machine i under schedule σ . Then the objective value is $C_{\text{envy}} = C_{\max}(\sigma) - C_{\min}(\sigma)$ for schedule σ . Moreover, we say a machine i is *extremal* under σ if $C_i^\sigma \in \{C_{\min}(\sigma), C_{\max}(\sigma)\}$. Let $A := \frac{p^\top \nu}{s^\top \mu}$ be the average (fractional) completion time of an instance. A machine i is *nice* if $p^\top \sigma_i \in [s_i A - p_{\max}; s_i A + p_{\max}]$, i.e. its load differs by at most $p_{\max}$ from its load on the fractional schedule. First, note the following simple property.

¹ For example, if one does binary search for OPT and introduces a variable (instead of a fixed value) T denoting the minimum completion time, i.e. all completion times have to be in $[T; T + \text{OPT}]$, then the largest coefficient does depend on $s_{\max}$. Due to this, ILP results like [30] do not yield the desired result (as we discussed with the authors).

Proposition 2. *For any schedule σ we either have $C_{\max}(\sigma) > A > C_{\min}(\sigma)$ or $C_{\max}(\sigma) = A = C_{\min}(\sigma)$.*

Proof. Observe that $\sum_{i \in [m]} s_i A = A \sum_{i \in [m]} s_i = \frac{p^\top v}{s^\top \mu} \sum_{i \in [m]} s_i = p^\top v$. We thus have $\sum_{i \in [m]} p^\top \sigma_i = \sum_{i \in [m]} s_i A \iff \sum_{i \in [m]} (s_i^{-1} p^\top \sigma_i - A) = \sum_{i \in [m]} (C_i^\sigma - A) = 0$. If one of the summands is non-zero, there has to be another non-zero summand of opposite sign. This shows the claim. $\square$

We can further show that an optimal schedule for C_{envy} always contains a machine that is both extremal and nice.

Lemma 13. *For $m > 1$, there is a schedule σ^* minimizing C_{envy} , i.e. minimizing $C_{\max}(\sigma^*) - C_{\min}(\sigma^*)$, such that an extremal machine is nice.*

Proof. Let σ^* be an optimal solution that minimizes the number of extremal machines. We proceed with proof by contradiction, assuming there is no extremal machine that is nice. Note that there have to be at least two extremal machines. Let i_1 such that $C_{\max}(\sigma^*) = C_{i_1}^{\sigma^*}$ and $i_2 \neq i_1$ such that $C_{\min}(\sigma^*) = C_{i_2}^{\sigma^*}$. By assumption, neither of those two is nice, i.e.

$$p^\top \sigma_{i_1}^* > s_{i_1} A + p_{\max} \quad \text{and} \quad p^\top \sigma_{i_2}^* < s_{i_2} A - p_{\max}.$$

We can thus move an arbitrary job from i_1 to i_2 without increasing $C_{\max}(\sigma^*)$ or decreasing $C_{\min}(\sigma^*)$. Thus, the solution does not become sub-optimal. By doing so, either the number of extremal machines is reduced, which contradicts that σ^* has a minimal number of extremal machines; or either i_1 or i_2 remain extremal, which contradicts the optimality of σ^* . $\square$

We can exploit this algorithmically by guessing the type of an extremal machine, guessing its load, which allows us to derive either the lower or upper bound on the completion times, and finally, finding the complementary bound via binary search.

Theorem 13. *There is an algorithm solving Q|HM(m, n)|C_{envy} in time $p_{\max}^{\mathcal{O}(d^2)} \cdot \text{poly}(\log n, \log m, \tau, d)$.*

Proof. We proceed as follows. Recall that the set of configurations $\mathcal{C}(\ell_t, u_t)$ is defined as $\mathcal{C}(\ell_t, u_t) = \{ c \in \mathbb{N}_{\geq 0}^d \mid \ell \leq p^\top c \leq u \}$.

1. Try the case $C_{\max}(\sigma) = C_{\min}(\sigma)$ by simply setting $\ell_t \leftarrow s_t A$, $u_t \leftarrow s_t A$, and setting up the corresponding [Configuration-ILP](#).
2. Try to solve this [Configuration-ILP](#) using [Lemma 12](#).
3. If successful, we have $\text{OPT} = 0$. Return the computed schedule.
4. If unsuccessful, then $\text{OPT} > 0$.

5. Guess the type t of a nice machine that is extremal, whose existence is ensured by [Lemma 13](#).
6. Guess the load of a nice extremal machine of type t , which is between $s_t A - p_{\max}$ and $s_t A + p_{\max}$ (at most $\mathcal{O}(p_{\max})$ possible values). As we have $C_{\max}(\sigma) > A > C_{\min}(\sigma)$, we thus know either $C_{\max}(\sigma)$ or $C_{\min}(\sigma)$.
7. Find the corresponding $C_{\min}(\sigma)$ or $C_{\max}(\sigma)$ value via binary search.
8. Set $\ell_t := \lceil C_{\min}(\sigma)s_t \rceil$ and $u_t := \lfloor C_{\max}(\sigma)s_t \rfloor$, set up the corresponding [Configuration-ILP](#).
9. Solve this [Configuration-ILP](#) using [Lemma 12](#).
10. For each of the guesses made, find the combination that has the best objective value and return the corresponding optimal schedule σ^* .

Since the guessing of a nice extremal machine of type t , and guessing its load, takes at most time $\mathcal{O}(\tau p_{\max})$, the run time is dominated by the time needed to solve each [Configuration-ILP](#). As clearly $\langle \ell_t, u_t \rangle = \text{poly}(\log n, \log m, \log p_{\max}, \tau, d)$ for all $t \in [\tau]$ for each instance of [Configuration-ILP](#) we try to solve with [Lemma 12](#), this yields an overall run time of $p_{\max}^{\mathcal{O}(d^2)} \cdot \text{poly}(\log n, \log m, \tau, d)$. $\square$

As $d \leq p_{\max}$, this implies that $Q|HM(m, n)|C_{\text{envy}}$ in FPT with parameter $p_{\max}$, which, to the best of our knowledge, was not known before.

4.4 ACHIEVING ETH-TIGHT RUN TIME FOR $Q|HM(m, n)|C_{\text{envy}}$

In this section, we give the $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$ time algorithm for solving $Q|HM(m, n)|C_{\text{envy}}$; this result represents an exponential speed-up w.r.t. to the parameters in contrast to the simpler algorithm in the previous section. As our results heavily rely on the ideas of Rohwedder [111] for solving $Q|HM(n)|C_{\max}$ in the same run time, we give a brief overview of these. The main idea is to partition the machines of an instance of $Q|HM(n)|C_{\max}$ into a set S of small ($\lfloor s_i T \rfloor \leq p_{\max}^4$) and big machines $B = \{1, \dots, m\} \setminus S$, for some guess on the makespan T . The set S then directly allows the use of small configurations. For the machines B , Rohwedder [111] proves that there exists a pivot element $a \in \{p_1, \dots, p_d\}$ such that there are optimal solutions which have at least $p_{\max}^2 |B|$ many jobs of this processing time scheduled on them. They then introduce an ILP dependent on a , called $\text{mod-IP}(a)$, where each machine $i \in S$ gets assigned “regular configurations” with load $L_i = \lfloor s_i T \rfloor$ (after the introduction of dummy jobs), but each machine

$i \in B$ gets assigned configurations with load $L_i \equiv_a \lfloor s_i T \rfloor$. At least $p_{\max}^2 |B|$ jobs of processing time a are not assigned to any machine.

mod-IP(a)

$$\begin{aligned}
\sum_{j=1}^n x_{ij} &= T_i & \forall i \in S \\
\sum_{j=1}^n x_{ij} &\equiv T_i \pmod{a} & \forall i \in B \\
\sum_{j:p_j=a} \sum_{i \in B} x_{ij} &\geq p_{\max}^2 \cdot |B| \\
\sum_{i=1}^m x_{ij} &= 1 & \forall j \in [n] \\
x_{ij} &\in \{0, 1\} & \forall j \in [n], i \in [m]
\end{aligned}$$

It is further shown that any solution to this ILP can be greedily extended to a solution of the original instance, which heavily relies on the fact that, before the $p_{\max}^2 |B|$ jobs of processing time a are assigned, there always exists a machine that has remaining capacity $> p_{\max}^2$. As each remaining machine capacity, as well as the number of jobs of each job type, are multiples of a , a greedy assignment of batches of size $ap_j \leq p_{\max}^2$ of each job type j to the machines is possible. The only remaining jobs of size a can also be greedily assigned to reach a load of exactly $\lfloor s_i T \rfloor$ on each machine i . Rohwedder [111] further shows that mod-IP(a) can be modeled as MULTI-CHOICE INTEGER PROGRAMMING:

MULTI-CHOICE INTEGER PROGRAMMING

Instance: $n, m, \Delta \in \mathbb{N}_{\geq 0}$, $A \in \mathbb{Z}^{m \times n}$ with $|A_{ij}| \leq \Delta$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$. Further, a partition P of $\{1, \dots, n\}$ and $t'_S \in \mathbb{N}_{\geq 0}$ for each $S \in P$.

Question: Which $x \in \mathbb{N}_{\geq 0}^n$ maximizes $c^T x$, s.t. $Ax = b$ and $\sum_{i \in S} x_i = t'_S$ for all $S \in P$?

This problem is shown to be solvable in time $(m\Delta|P|)^{\mathcal{O}(m)}(n + t')^{\mathcal{O}(1)}$ [111, Theorem 2], where $t' = \sum_{S \in P} t'_S$, and thus in $p_{\max}^{\mathcal{O}(d)} n^{\mathcal{O}(1)}$ when inserting the respective ILP parameters of mod-IP(a), resulting in the run time of $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$ for $Q|HM(n)|C_{\max}$. MULTI-CHOICE INTEGER PROGRAMMING can also be reduced in polynomial time to a *Combinatorial N -fold ILP*, which is a class of N -fold ILPs with $B^{(1)} = B^{(N)} = \mathbb{1}$ and $S = 1$. There exist some recent results regarding the efficient parameterized solving of such ILPs [61, 75], which will be useful later in this section.

We proceed similarly to Rohwedder [111]. Given an instance I of $Q|HM(m, n)|C_{\text{envy}}$, we start by guessing the values for $C_{\max}$ and $C_{\min}$

of an optimal solution, which is possible in time $\mathcal{O}(\tau p_{\max})$ as described in [Theorem 13](#); we can thus now assume to know these values, and any solution σ for I with $\lfloor s_i C_{\min} \rfloor \leq \sum_{j \in [d]} \sigma_j(i) \leq \lfloor s_i C_{\max} \rfloor$ for all $i \in [m]$ is an optimal solution for I . We also partition the m machines into a set of small machines $S = \{i : \lfloor s_i C_{\max} \rfloor < p_{\max}^4 + 4p_{\max}\}$ and big machines $B = \{1, \dots, m\} \setminus S$, but also consider a further subset of B with a large discrepancy between $\lfloor s_i C_{\max} \rfloor$ and $\lceil s_i C_{\min} \rceil$, namely $F := \{i \in B : \lfloor s_i C_{\max} \rfloor - \lceil s_i C_{\min} \rceil \geq 4p_{\max}\}$. While, for easier readability, we here talk about single machines i in S, B, F , we still assume the machines to be encoded in high-multiplicity. So when considering some machine $i \in S \cup B$, we actually consider all machines of type $t \in [\tau]$, where t is the type of machine i . We prove the following statement for F :

Lemma 14. *For any instance I of $Q|HM(m, n)|C_{\text{envy}}$, there exists an optimal schedule σ^* that satisfies at least one of the following properties.*

- (i) $L_i > \lfloor s_i C_{\max} \rfloor - 2p_{\max}, \forall i \in F$
- (ii) $L_i < \lceil s_i C_{\min} \rceil + 2p_{\max}, \forall i \in F$
- (iii) $\sum_{i \in F} (\lceil s_i C_{\min} \rceil + p_{\max}) \leq \sum_{i \in F} L_i \leq \sum_{i \in F} (\lfloor s_i C_{\max} \rfloor - p_{\max})$

Proof. Assume, (iii) does not hold. Then either $\sum_{i \in F} L_i > \sum_{i \in F} (\lfloor s_i C_{\max} \rfloor - p_{\max})$, or $\sum_{i \in F} L_i < \sum_{i \in F} (\lceil s_i C_{\min} \rceil + p_{\max})$.

Case 1: $\sum_{i \in F} L_i > \sum_{i \in F} (\lfloor s_i C_{\max} \rfloor - p_{\max})$. In this case we show that either (i) holds, or we transform the given optimal schedule such that it does. Suppose (i) does not hold. Then, there always exists a machine $i \in F$ with $L_i \leq \lfloor s_i C_{\max} \rfloor - 2p_{\max}$. By assumption, there must exist a machine $i' \in F$ with $L_{i'} > \lfloor s_{i'} C_{\max} \rfloor - p_{\max}$. We may now move any job from i' to i without changing the objective value of the solution. As this does not change the total load of machines in F , we can apply this transformation iteratively. By the definition of F , the load on i' stays above $s_{i'} \lceil C_{\min} \rceil$. Let $L'_{i'}$ be the load of the machine i' after we moved the job. Then, we have $L'_{i'} \geq L_{i'} - p_{\max} > \lfloor s_{i'} C_{\max} \rfloor - 2p_{\max} \geq \lceil s_{i'} C_{\min} \rceil + 2p_{\max}$, i. e. after all iterations, only (i) is fulfilled.

Case 2: $\sum_{i \in F} L_i < \sum_{i \in F} (\lceil s_i C_{\min} \rceil + p_{\max})$. In this case we show that either (ii) holds, or we transform the given optimal schedule such that it does. Suppose (ii) does not hold. Then, there exists a machine $i \in F$ with $L_i \geq \lceil s_i C_{\min} \rceil + 2p_{\max}$. By assumption, there must exist a machine $i' \in F$ with $L_{i'} < \lceil s_{i'} C_{\min} \rceil + p_{\max}$. We may now move any job from i' to i without changing the objective value of the solution. As before, we can apply this transformation iteratively.

Thus, in any case, one of the three properties must hold. $\square$

Again, similar to the idea of Rohwedder [111] concerning a pivot element $a \in \{p_1, \dots, p_d\}$, we prove the following statement.

Lemma 15. *For any instance I of $Q|HM(m, n)|C_{\text{envy}}$ and any optimal schedule σ^* of I , there are at least $p_{\max}^2|B|$ jobs of size p_j scheduled on machines in B , for some job type $j \in [d]$.*

Proof. Let i be any machine in $B \setminus F$, i' any machine in F . Recall that $\lfloor s_i C_{\max} \rfloor \geq p_{\max}^4 + 4p_{\max}$ for all i by definition. Furthermore, $\lfloor s_i C_{\max} \rfloor - \lceil s_i C_{\min} \rceil < 4p_{\max}$, and thus $\lceil s_i C_{\min} \rceil > p_{\max}^4$, for all i . As $\lceil s_{i'} C_{\min} \rceil \geq \lceil s_i C_{\min} \rceil$ for all i' and i by definition of F , $\lceil s_{i''} C_{\min} \rceil > p_{\max}^4$ for all machines $i'' \in B$. As $p_j \leq p_{\max}$ for all $j \in [d]$, there must be at least $p_{\max}^3$ jobs on each of the $|B|$ machines in B , i. e. at least $p_{\max}^3|B|$ in total. As there are only $d \leq p_{\max}$ many different job types, by pigeon-hole, there must exist some $j \in [d]$ such that there are at least $p_{\max}^2|B|$ jobs of type j on machines in B . $\square$

Let a be the processing time of a job type for which Lemma 15 holds; we also call this the pivot element a , and can guess a in at most $\mathcal{O}(d)$ guesses. We also guess which of the properties of Lemma 14 is true for the machines in F in at most 3 guesses. Depending on this guess, we set

$$T_i := \begin{cases} \lfloor s_i \lfloor C_{\max} \rfloor \rfloor & \text{if } i \notin F \\ \lfloor s_i \lfloor C_{\max} \rfloor \rfloor & \text{if } i \in F \text{ and Lemma 14, (i) or (iii) holds} \\ \lfloor s_i \lceil C_{\min} \rceil \rceil + 2p_{\max} & \text{if } i \in F \text{ and Lemma 14, (ii) holds} \end{cases}$$

We now aim to create a configuration-based ILP similar to Rohwedder [111] which is not only solvable in the target time of $p_{\max}^{\mathcal{O}(d)}|I|^{\mathcal{O}(1)}$, but whose solution we can transform to a solution of $Q|HM(m, n)|C_{\text{envy}}$ in even less time. We first create a “dummy job type” $d+1$ with $p_{d+1} = 1$, and set $v_{d+1} = \sum T_i - \sum p_j$ to be able to always find solutions where each machine i has load exactly T_i . For the problem of makespan minimization, arbitrary placement of dummy jobs is not a problem as long as the target load T_i is hit. However for C_{envy} , we have to guarantee that the actual load on each machine (without dummy jobs) is above $s_i \lceil C_{\min} \rceil$. We want to do this by carefully constructing the configurations on each machine type, and not placing dummy jobs by any other means if possible. We define the the set of configurations $\mathcal{C}_i$ for each machine $i \in [m]$ as follows:

$$\mathcal{C}_i := \{c \in \mathbb{Z}_{\geq 0}^{d+1} : c_{d+1} + \sum_{j=1}^d p_j c_j = T_i, \\ c_{d+1} \leq T_i - \lceil s_i C_{\min} \rceil \leq p_{\max}^4 + 4p_{\max}\}$$

for $i \in S$,

$$\mathcal{C}_i := \{c \in \mathbb{Z}_{\geq 0}^{d+1} : c_{d+1} + \sum_{j=1}^d p_j c_j \equiv_a T_i, c_j < a, \forall j \in [d], \\ c_{d+1} \leq T_i - \lceil s_i C_{\min} \rceil \leq 4p_{\max}\}$$

for $i \in B \setminus F$, and

$$\mathcal{C}_i := \{c \in \mathbb{Z}_{\geq 0}^{d+1} : c_{d+1} + \sum_{j=1}^d p_j c_j \equiv_a T_i, c_j < a, \forall j \in [d], \\ c_{d+1} \leq \max(T_i - \lceil s_i C_{\min} \rceil, 2p_{\max})\}$$

for $i \in F$.

This (re-)definition is different from and independent of the definition of configurations w. r. t. lower and upper bounds in [Section 1.1](#). Note that, while the configurations are defined separately for each machine i , there is in fact only a limited number of different configuration sets $\mathcal{C}_i$ in total: For all machines $i, i' \in S$ with $T_i = T_{i'}$ and $T_i - \lceil s_i C_{\min} \rceil = T_{i'} - \lceil s_{i'} C_{\min} \rceil$, $\mathcal{C}_i = \mathcal{C}_{i'}$, i. e. the machines have the same *machine type* w. r. t. the set of configurations. The equality $\mathcal{C}_i = \mathcal{C}_{i'}$ also holds for $i, i' \in B \setminus F$, when $T_i \equiv_a T_{i'}$ and $T_i - \lceil s_i C_{\min} \rceil = T_{i'} - \lceil s_{i'} C_{\min} \rceil$, and for $i, i' \in F$ where $T_i \equiv_a T_{i'}$. For $i \in S$, there are at most $p_{\max}^{O(1)}$ different machine types, as both $T_i \leq p_{\max}^4 + 4p_{\max}$ and $T_i - \lceil s_i C_{\min} \rceil \leq p_{\max}^4 + 4p_{\max}$; for $i \in B \setminus F$ as well as $i \in F$, there are at most a different values for $T_i \bmod a$, implying at most $a + 2p_{\max}$ different machine types in F , and at most $a + 4p_{\max}$ different machines types in $B \setminus F$, as $T_i - \lceil s_i C_{\min} \rceil \leq 4p_{\max}$ for all $i \in B \setminus F$. Thus, the total number of different machine types is at most $p_{\max}^{O(1)}$; we call this number τ' , and define the number of machines with the same type as μ_t , for each $t \in [\tau']$. Also note that $|c_j| \leq p_{\max}^4 + 4p_{\max}$ for all $j \in [d]$ and all $c \in \mathcal{C}_t$, for any $t \in [\tau']$, i. e. the biggest entry in any configuration is at most $p_{\max}^4 + 4p_{\max}$. Consider the following ILP that depends on the configurations and the pivot element a .

Conf-Mod-IP(a)

$$\sum_{c \in \mathcal{C}_t} y_{t,c} = \mu_t \quad \forall t \in [\tau'] \quad (4.1)$$

$$\sum_{t \in [\tau']} \sum_{c \in \mathcal{C}_t} c_j y_{t,c} = v_j - ax_j \quad \forall j \in [d+1], p_j \neq a \quad (4.2)$$

$$\sum_{t \in [\tau']} \sum_{c \in \mathcal{C}_t} c_j y_{t,c} \leq v_j - p_{\max}^2 |B| \quad \forall j \in [d], p_j = a \quad (4.3)$$

$$\sum_{t \in [\tau']} \sum_{c \in \mathcal{C}_t} c_{d+1} y_{t,c} = v_{d+1} \quad (4.4)$$

with variables

$$y_{t,c} \in \mathbb{N}_{\geq 0} \quad \forall t \in [\tau'], c \in \mathcal{C}_t$$

$$x_j \in \mathbb{N}_{\geq 0} \quad \forall j \in [d]$$

Introducing the x variables is similar to the idea of slack variables by Jansen et al. [75] for solving $Q|HM(n)|C_{\min}$, and will help us to keep all job quantities as multiples of a after solving the ILP. We first show

that, if there exists a feasible solution to $Q|HM(m, n)|C_{\text{envy}}$, there also exists a feasible solution to $\text{Conf-Mod-IP}(a)$, or at least a slight variation thereof. Let $\text{Conf-Mod-IP}(a)'$ be the above ILP where [Constraint 4.4](#) is replaced by $\sum_{t \in [\tau'_F]} \sum_{c_{d+1} \in \mathcal{C}_t} c_{d+1} y_{t,c} = b$ for some $b \in \mathbb{N}_{\geq 0}$, where τ'_F is the number of machine types for machines $i \in S \cup (B \setminus F)$. We hereby implicitly assume that numbers $1, \dots, \tau'_F$ represent the types of machines in $S \cup (B \setminus F)$. The number b represents the total number of dummy jobs on machines in $S \cup (B \setminus F)$ in a solution y . In the following, let L_i be the load of machine i in a solution to $Q|HM(m, n)|C_{\text{envy}}$.

Lemma 16. *If there exists a feasible solution σ^* to $Q|HM(m, n)|C_{\text{envy}}$ and either [Lemma 14](#), (i) or (ii) holds, there exists a feasible solution to $\text{Conf-Mod-IP}(a)$. If [Lemma 14](#), (iii) holds instead, there exists a feasible solution to $\text{Conf-Mod-IP}(a)'$ with $b = \sum_{i \in S \cup (B \setminus F)} (T_i - L_i)$.*

Proof. Let σ^* be a feasible solution to $Q|HM(m, n)|C_{\text{envy}}$ for which one of the properties of [Lemma 14](#) holds. We set T_i according to which property holds, as defined before. We show the statement by constructing a solution (y, x) to either $\text{Conf-Mod-IP}(a)$ or $\text{Conf-Mod-IP}(a)'$ from σ^* . At the start, we set $y_{t,c} = 0$ for all $t \in [\tau']$ and $c \in \mathcal{C}_t$, and $x_j = 0$ for all $j \in [d]$. By [Lemma 15](#), we know that for any optimal solution σ^* to $Q|HM(m, n)|C_{\text{envy}}$ there exists a job type $j \in [d]$ with $p_j = a$ such that at most $v_j - p_{\max}^2 |B|$ jobs of this type are assigned to machines in S . We also know that $L_i \leq T_i$ and $L_i \geq \lceil s_i C_{\min} \rceil$ for all $i \in [m]$ by the optimality of σ^* and the respective property of [Lemma 14](#).

By construction of the configurations, for all machines $i \in S$ of some type $t \in [\tau']$, there exists a $c \in \mathcal{C}_t$ with $\sum_{j=1}^d p_j c_j = L_i$ and $c_{d+1} = T_i - L_i$, such that the total number of jobs of type j with $p_j = a$ over all such c is at most $v_j - p_{\max}^2 |B|$. For each $i \in S$, we select such c by setting $y_{t,c} = y_{t,c} + 1$. Recall that $\sigma_j^*(i)$ is the number of jobs of type j assigned to machine i by σ^* . Similarly, for all machines $i \in B$, there exists a configuration in $c \in \mathcal{C}_t$ of some type $t \in [\tau']$ with $\sum_{j=1}^d p_j c_j \equiv_a L_i$ and $c_j \equiv_a \sigma_j^*(i)$, as, for any schedule with $\sigma_j^*(i) > a$ for any $j \in [d]$, there exists a configuration with $c_j < a$ and equivalent parity modulo a . This implies there exists some $k_j \in \mathbb{N}_{\geq 0}$ for each $j \in [d]$, such that $c_j = \sigma_j^*(i) - a \cdot k_j < a$. Moreover, there exists such c with $c_{d+1} = T_i - L_i \leq 4p_{\max}$ for all $i \in B \setminus F$. If either [Lemma 14](#), (i) or (ii) holds for σ^* , such c exist for machines $i \in F$ with $c_{d+1} = T_i - L_i \leq 2p_{\max}$; if not, there exist c with $c_{d+1} \equiv_a T_i - L_i$. We select c with these properties for all $i \in B$ by setting $y_{t,c} = y_{t,c} + 1$, and set $x_j = x_j + k_j$ for each $j \in [d]$.

Now, we have selected a configuration for each machine (or, more specifically, μ_t for each machine type), and thus [Constraint 4.1](#) is fulfilled. The configurations y are chosen in such a way that exactly $v_j - ax_j$ are assigned over all machines for each $j \in [d]$, fulfilling [Constraint 4.2](#) for all job types but $d + 1$. By the same argument, none

of the jobs j with $p_j = a$ are assigned on machines in B ; as there are at least $p_{\max}^2 |B|$ jobs of this type assigned on machines in B in σ^* by choice of σ^* , [Constraint 4.3](#) is fulfilled.

If either [Lemma 14](#), (i) or (ii) holds for σ^* , we have selected configurations c with $c_{d+1} = T_i - L_i$ for all machines $i \in F$, and thus configurations c with $c_{d+1} = T_i - L_i$ for all machines $i \in [m]$. As we have introduced exactly $v_{d+1} = \sum T_i - \sum p_j = \sum T_i - \sum L_i$ many dummy jobs of size $p_{d+1} = 1$, this implies that [Constraint 4.4](#) is valid for y by our selection of c 's for y . This specifically means that here, all dummy jobs are assigned via the configurations. We set $x_{d+1} = 0$, fulfilling [Constraint 4.2](#) for $d + 1$ as well, and thus all constraints. This implies that (y, x) is a feasible solution to [Conf-Mod-IP\(a\)](#).

If [Lemma 14](#), (iii) holds instead, it may be that $T_i - L_i > 2p_{\max}$ for machines in $i \in F$, and $c_{d+1} \leq 2p_{\max} < T_i - L_i$ for the configuration c assigned to i by y . We thus replace [Constraint 4.4](#) by $\sum_{t \in [\tau'_F]} \sum_{c_{d+1} \in \mathcal{C}_t} c_{d+1} y_{t,c} = b$ for $b = \sum_{i \in S \cup (B \setminus F)} \sigma_{d+1}^*(i)$. This constraint is fulfilled immediately by our choice of configurations for machines in S and $B \setminus F$. There still may be $\sum T_i - \sum p_j - (b + \sum_{i \in F} c_{d+1})$ dummy jobs not assigned by y . However, again by our choice of configurations, $\sum T_i - \sum p_j - (b + \sum_{i \in F} c_{d+1})$ must be a multiple of a , and thus $\sum T_i - \sum p_j - (b + \sum_{i \in F} c_{d+1}) = k \cdot a$ for some $k \in \mathbb{N}_{\geq 0}$. We set $x_{d+1} = k$, fulfilling [Constraint 4.2](#) for $d + 1$ as well, and thus all constraints. This implies that (y, x) is a feasible solution to [Conf-Mod-IP\(a\)](#)'. $\square$

In any case, [Conf-Mod-IP\(a\)](#) and [Conf-Mod-IP\(a\)](#)' both have the structure of *Combinatorial N-fold ILP*. Such ILPs can be solved in time $(NR\Delta)^{\mathcal{O}(R)} \log(\|b'\|_{\infty})$ [[75](#), Theorem 1], where b' is the RHS of the ILP. In case of [Conf-Mod-IP\(a\)](#) and [Conf-Mod-IP\(a\)](#)', the relevant N -fold parameters are $N = \tau' + d + 1 \in p_{\max}^{\mathcal{O}(1)}$, $R \in \mathcal{O}(d)$, $\Delta \leq p_{\max}^4 + 4p_{\max}$, and $\log(\|b'\|_{\infty}) \in |I|^{\mathcal{O}(1)}$. Both ILPs can thus be solved in time $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$ by the algorithm of Jansen et al. [[75](#), Theorem 1]. Now, we show how to greedily augment a solution to [Conf-Mod-IP\(a\)](#) to a solution of $Q|HM(m, n)|C_{\text{envy}}$.

Lemma 17. *Given a feasible solution to [Conf-Mod-IP\(a\)](#) or [Conf-Mod-IP\(a\)](#)', a solution to $Q|HM(m, n)|C_{\text{envy}}$ can be computed in time $p_{\max}^{\mathcal{O}(1)} |I|^{\mathcal{O}(1)}$.*

Proof. Let (y, x) be a solution to [Conf-Mod-IP\(a\)](#) or [Conf-Mod-IP\(a\)](#)'. We compute a feasible solution σ^* to $Q|HM(m, n)|C_{\text{envy}}$ from (y, x) . We know that [Conf-Mod-IP\(a\)](#)' is valid for $b = \sum_{i \in S \cup (B \setminus F)} (T_i - L_i)$, where L_i is the load of machine i in σ^* . We assume we are given such a solution to [Conf-Mod-IP\(a\)](#)'. Let L'_i the current load of i in σ^* at some step in our assignment process. Any solution y already assigns exactly $T_i = \lfloor s_i C_{\max} \rfloor$ load to machines $i \in S$. We thus simply set $\sigma_i^*(j) = c_j$ for all $j \in [d + 1]$ and $i \in S$. As $c_{d+1} \leq T_i - \lceil s_i C_{\min} \rceil \leq \lfloor s_i C_{\max} \rfloor - \lceil s_i C_{\min} \rceil$ for the configurations $\mathcal{C}_i$ of such machines, this implies that $\lceil s_i C_{\min} \rceil \leq L'_i \leq \lfloor s_i C_{\max} \rfloor$ for all $i \in S$ even after the removal of dummy jobs. We proceed by assigning the load of configurations to machines $i \in B$,

i. e. set $\sigma_i^*(j) = c_j$ for all $j \in [d+1]$ and $i \in B$. By construction of the configurations C_i for $B \setminus F$ and F and the constraints of the respective ILPs, $T_i - L'_i = a \cdot k$ for some $k \in \mathbb{N}_{\geq 0}$ and all $i \in B$, i. e. the missing load after assignment of all jobs from their respective configurations on each big machine is a multiple of a .

There exist at most $\tau \cdot \tau' = \tau p_{\max}^{O(1)}$ many different values $T_i - L'_i$: There are at most τ' different configurations assigning different loads L'_i to the machines, while T_i depends only on the type $t \in [\tau]$ of each machine $i \in B$. We define the number of different “load types”, i. e. all different $T_i - L'_i$ values, to be $\tau^L \in \tau p_{\max}^{O(1)}$, and μ^L to encode the multiplicities of all types $t \in \tau^L$, i. e. μ_t^L returns the number of machines with the same load type in $[\tau^L]$.

By [Constraint 4.2](#), the total load of unassigned jobs of each type $j \in [d+1]$, $p_j \neq a$ is a multiple of a as well. Furthermore, no jobs of size a are placed yet on big machines at all, and we know that for any optimal solution σ^* to $Q|HM(m, n)|C_{\text{envy}}$, there are at least $p_{\max}^2 |B|$ jobs of size a placed on the big machines by [Lemma 15](#). Thus, while no jobs of processing time a are placed on machines in B , by pigeon-hole principle, there must always exist a machine $i \in B$ with current load $L'_i \leq T_i - p_{\max}^2$. This machine i is of some load type $t \in [\tau^L]$, i. e. all μ_t^L machines of load type t have the same missing load $T_i - L'_i$. We now first place all jobs of job types j with $p_j \neq a$, and afterwards all jobs with processing time a .

We again first assume that either [Lemma 14](#), (i) or (ii) holds, i. e. (y, x) is a solution to [Conf-Mod-IP\(a\)](#) and all dummy jobs are already assigned by y ([Constraint 4.4](#)). Thus, the only jobs not yet assigned are of types $j \in [d]$. Let v'_j be the number of yet unassigned jobs of type j . While there is still a number $v'_j \leq x_j$ of unassigned jobs of a type j with $p_j \neq a$, we take select some load type $t \in [\tau^L]$ with current load $L'_t \leq T_t - p_{\max}^2$ in σ^* , which we know to exist by the pigeon-hole argument. We calculate the maximum k such that $k \cdot a \leq v'_j$ and $L'_t + p_j(k \cdot a) \leq T_t$. If $v'_j < ka\mu_t^L$, we assign ka jobs of type j each to a number $\mu_t^{L'} = \left\lfloor \frac{v'_j}{ka} \right\rfloor$ of machines of load type t (encoding this assignment in high-multiplicity), and $(v'_j \bmod ka)$ jobs to 1 machine of load type t , leaving the remaining $\mu_t^L - \mu_t^{L'} - 1$ machines of type t untouched. If instead $v'_j \geq ka\mu_t^L$, we assign ka jobs of type j each to all μ_t^L machines of load type t . In the former case, we now have 3 possible different assignments for machines of type t . We deal with this by adjusting the number of load types τ^L , incorporating the machines that were assigned ka jobs each as a new type t' into τ^L , with $\mu_{t'}^L = \mu_t^{L'}$, the single machine that was assigned $v'_j \bmod ka$ jobs as a new type t'' with $\mu_{t''}^L = 1$, and setting $\mu_t^L = \mu_t^L - \mu_t^{L'} - 1$. In the latter case, all machines of type t now have load type $T'_t - L'_t + kap_j$, i. e. are all still of the same load type. We iterate this procedure until $v'_j = 0$. For each job type j , the number of different load types is increased by at most

2: Only if all jobs of type j are placed on some machines of load type t , i. e. $v'_j = 0$ after the current step, the number of load types increases by 2. So, the number of load types is increased by at most $2d$ when iterating over all job types. This process then places all job types in time $\mathcal{O}(\tau^L d)$, as we consider each load type at most once for each job type. As argued, $\tau^L \in \tau p_{\max}^{\mathcal{O}(1)} + 2d$ in each iteration, which implies a run time of $p_{\max}^{\mathcal{O}(1)} |I|^{\mathcal{O}(1)}$ for the whole process.

Now, the only jobs left to place are of type j with $p_j = a$. As so far we have only placed job bundles whose size is a multiple of a , the missing load on each big machine until T_i is reached is still a multiple of a . On any machine of load type $t \in \tau^L$ with $T_t - L'_t > 0$, all μ_t^L machines are assigned $(T_t - L'_t)/a$ many jobs of processing time a in σ^* ; this assignment is encoded in high-multiplicity, and thus possible in time $\mathcal{O}(\tau^L) \in p_{\max}^{\mathcal{O}(1)} |I|^{\mathcal{O}(1)}$. This assignment must be possible due to the fact that $\sum T_i = \sum p_j$ after the introduction of dummy jobs. Now, all jobs of types $j \in [d+1]$ are placed, and all machines have load exactly T_i . Here, dummy jobs have only been assigned via the configurations. Each configuration assigns at most $T_i - \lceil s_i C_{\min} \rceil$ dummy jobs on each i by, either simply by definition of configurations for machines in S and $B \setminus F$, or with the additional fact that $\max(T_i - \lceil s_i C_{\min} \rceil, 2p_{\max}) = T_i - \lceil s_i C_{\min} \rceil$ for $i \in F$ when either Lemma 14, (i) or (ii) holds. Thus, $\lceil s_i C_{\min} \rceil \leq L'_i \leq T_i \leq \lfloor s_i C_{\max} \rfloor$ for all $i \in [m]$ after the removal of dummy jobs. The schedule σ^* is therefore a feasible solution for $Q|HM(m, n)|C_{\text{envy}}$ after the removal of all dummy jobs. The number of different job assignments in σ^* is at most $\tau^L \in \tau p_{\max}^{\mathcal{O}(1)} + 2d$, and can thus be output in time $p_{\max}^{\mathcal{O}(1)} |I|^{\mathcal{O}(1)}$.

Now, instead assume that Lemma 14, (iii) holds, i. e. (y, x) is a solution to Conf-Mod-IP(a)' and not all dummy jobs are assigned by y . Let v'_j again be the as of yet unassigned jobs of type j . By our assignment and the definition of dummy jobs, there are at most $v'_{d+1} \leq \sum T_i - \sum p_j - b$ unassigned dummy jobs remaining. We know that it must hold for the remaining dummy jobs that $v'_{d+1} \leq \sum T_i - \sum p_j - b = \sum T_i - \sum p_j - \sum_{i \in S \cup (B \setminus F)} (T_i - L_i) = \sum_{i \in F} (T_i - L_i)$ by choice of b and the number of introduced dummy jobs. As for now, we have only assigned jobs according to y , we know that v'_{d+1} is divisible by a due to Constraint 4.2. For each load type $t \in \tau^L$ with $t \in F$ that is assigned some configuration c by y , we calculate the maximum k such that $k \cdot a \leq v'_{d+1}$ and $L'_t + p_j(k \cdot a) \leq T_t - \lceil s_t C_{\min} \rceil$. Again, if $v'_{d+1} < ka\mu_t^L$, we assign ka jobs of type j each to a number $\mu_t^{L'} = \lfloor \frac{v'_j}{ka} \rfloor$ of machines of load type t (encoding this assignment in high-multiplicity), and $(v'_j \bmod ka)$ jobs to 1 machine of load type t , leaving the remaining $\mu_t^{L'} - \mu_t^{L'} - 1$ machines of type t untouched; if instead $v'_{d+1} \geq ka\mu_t^L$, we assign ka jobs of type j each to all μ_t^L machines of load type t . We again adjust the number of load types τ^L accordingly. We iterate this

procedure until $v'_{d+1} = 0$. This is always possible: By [Lemma 14](#), (iii), $\sum_{i \in F} (\lceil s_i C_{\min} \rceil + p_{\max}) \leq \sum_{i \in F} L_i \leq \sum_{i \in F} (\lfloor s_i C_{\max} \rfloor - p_{\max})$. Thus,

$$v'_{d+1} \leq \sum_{i \in F} (T_i - L_i) \leq \sum_{i \in F} T_i - (\lceil s_i C_{\min} \rceil + p_{\max}) .$$

As $a \leq p_{\max}$, at least $T_i - \lceil s_i C_{\min} \rceil - p_{\max}$ dummy jobs per machine $i \in F$, and thus at least $\sum_{i \in F} T_i - \lceil s_i C_{\min} \rceil - p_{\max}$ dummy jobs in total can be assigned on machines in F , allowing all remaining v'_{d+1} dummy jobs to be assigned in multiples of a . As argued before, this process takes at most $\mathcal{O}(\tau^L) \in p_{\max}^{\mathcal{O}(1)} |I|^{\mathcal{O}(1)}$ steps even after incorporation of the at most $2d$ new load types in F .

Now, the only jobs not yet assigned are of types $j \in [d]$. As so far we have only placed job bundles whose size is a multiple of a , the missing load on each big machine until T_i is reached is still a multiple of a . We continue exactly as before (the case when [Lemma 14](#), (i) or (ii) holds): We first place all jobs of types $j \in [d]$ with $p_j \neq a$ in time $\mathcal{O}(\tau^L) \in p_{\max}^{\mathcal{O}(1)} |I|^{\mathcal{O}(1)}$, and afterwards, $(T_i - L'_i)/a$ many jobs of processing time a each on machines of each load type $t \in \tau^L$ in σ^* in the same run time. This again must be possible due to the fact that $\sum T_i = \sum p_j$ after the introduction of dummy jobs. Now, all jobs of types $j \in [d+1]$ are placed, and all machines have load exactly T_i . By construction of $\mathcal{C}_i$ for $i \in B \setminus F$ and the greedy assignment of dummy jobs only onto machines in F , $\lceil s_i C_{\min} \rceil \leq L'_i \leq T_i = \lfloor s_i C_{\max} \rfloor$ holds for all $i \in B \setminus F$ after the removal of dummy jobs. On each machine $i \in F$, we have placed at most $T_i - \lceil s_i C_{\min} \rceil$ dummy jobs in total by configurations and greedy assignment, implying that $\lceil s_i C_{\min} \rceil \leq L'_i \leq T_i \leq \lfloor s_i C_{\max} \rfloor$ after the removal of dummy jobs. The schedule σ^* is therefore a feasible solution for $Q|HM(m, n)|C_{\text{envy}}$ after the removal of all dummy jobs in this case as well. The number of different job assignments in σ^* is at most $\tau^L \in \tau p_{\max}^{\mathcal{O}(1)} + \mathcal{O}(d)$, and can thus be output in time $p_{\max}^{\mathcal{O}(1)} |I|^{\mathcal{O}(1)}$. $\square$

We now unify all these results to obtain the following theorem.

Theorem 14. *There is an algorithm solving $Q|HM(m, n)|C_{\text{envy}}$ in time $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$.*

Proof. Guess the values for $C_{\max}$ and $C_{\min}$ in an optimal solution in time $\mathcal{O}(\tau p_{\max}) \in \mathcal{O}(p_{\max}) |I|^{\mathcal{O}(1)}$ as described in [Theorem 13](#). Guess which case of [Lemma 14](#) holds for the given instance I of $Q|HM(m, n)|C_{\text{envy}}$ in time $\mathcal{O}(1)$. Construct and solve the respective instance of [Conf-Mod-IP\(a\)](#), or [Conf-Mod-IP\(a\)'](#) with $b = \sum_{i \in \text{SU}(B \setminus F)} (T_i - L_i)$ as a Configuration N -fold ILP, in time $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$ [[75](#), Theorem 1]; let (y, x) be the computed solution. Compute a feasible solution to I from (y, x) with [Lemma 17](#) in time $p_{\max}^{\mathcal{O}(1)} |I|^{\mathcal{O}(1)}$.

We do not know the value $b = \sum_{i \in \text{SU}(B \setminus F)} (T_i - L_i)$, however, we can guess it in at most $\log(mp_{\max}^{\mathcal{O}(1)})$ guesses: As $0 \leq \sum_{i \in \text{SU}(B \setminus F)} (T_i - L_i) \leq$

$\sum_{i \in S} (p_{\max}^4 + 4p_{\max}) + \sum_{i \in B \setminus F} 4p_{\max}$ and $|S \cup B \setminus F| \leq m$, there are at most $mp_{\max}^{\mathcal{O}(1)}$ different possible values for b . If, for some given b , [Conf-Mod-IP\(a\)'](#) does not return a feasible solution, we know that we have to decrease b , as in this case, too many dummy jobs were placed on machines in $S \cup (B \setminus F)$, i. e. $b < \sum_{i \in S \cup (B \setminus F)} (T_i - L_i)$. On the other hand, if [Conf-Mod-IP\(a\)'](#) does return a feasible solution, but the greedy strategy fails, this can only be due to the fact that too many dummy jobs were placed on machines in F , i. e. $b > \sum_{i \in S \cup (B \setminus F)} (T_i - L_i)$, and we have to increase b . This implies that we can guess the correct value for b via Binary Search on the set $\{0, \dots, \sum_{i \in S} (p_{\max}^4 + 4p_{\max}) + \sum_{i \in B \setminus F} 4p_{\max}\}$ in at most $\log(mp_{\max}^{\mathcal{O}(1)})$ guesses, increasing the run time by a factor at most $\log(mp_{\max}^{\mathcal{O}(1)}) \in |I|^{\mathcal{O}(1)}$. All these steps bring the run time up to total of $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$, concluding the proof. $\square$

This run time is, in fact, the asymptotically best possible run time for these parameters under the ETH: Jansen et al. [76] have shown that $P||C_{\max}$, and thus $Q|HM(m, n)|C_{\max}$, cannot be solved in time $p_{\max}^{\mathcal{O}(d^{1-\delta})}$ for any $\delta > 0$ under the ETH. Any instance I of $Q|HM(m, n)|C_{\max}$ can easily be reduced to an instance I' of $Q|HM(m, n)|C_{\text{envy}}$ by the introduction of $\sum [s_i T] - \sum p_j$ dummy jobs of processing time 1 for some makespan guess T , such that I has a feasible solution with makespan $\leq T$ only if I' has a solution with objective value $C_{\text{envy}} = 0$. This shows that $Q|HM(m, n)|C_{\text{envy}}$ cannot be solved in time $p_{\max}^{\mathcal{O}(d^{1-\delta})}$ for any $\delta > 0$ under the ETH.

4.5 CONCLUSION

Firstly, we showed that the problem $Q||C_{\text{envy}}$ is FPT in $p_{\max}$ alone by combining results from the literature with a novel binary search routine on the objective value of this problem. We in fact managed to prove this result by giving an algorithm with run time $p_{\max}^{\mathcal{O}(d^2)} \text{poly}(\log n, \log m, \tau, d)$ for $Q|HM(m, n)|C_{\text{envy}}$, matching the run times of algorithms for $Q|HM(m, n)|C_{\max}$ and $Q|HM(m, n)|C_{\min}$. Secondly, we managed to improve the run time for solving $Q|HM(m, n)|C_{\text{envy}}$ to $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$, matching the run time for solving $Q|HM(n)|C_{\max}$ [111] and $Q|HM(n)|C_{\min}$ [75].

In summary, for parameters $p_{\max}$ and d , there now exist algorithms for scheduling on uniform machines of $HM(n)$ encoded instances with asymptotically best possible run times for these parameters under the ETH, namely $p_{\max}^{\mathcal{O}(d)} |I|^{\mathcal{O}(1)}$, for objectives $C_{\max}$, $C_{\min}$ and C_{envy} . In fact, our algorithm also solves $HM(m, n)$ encoded instances of $Q||C_{\text{envy}}$ in the given run time. It is only natural to ask if the ideas presented here can also be applied to $HM(m, n)$ encoded instances of $Q||C_{\max}$ and $Q||C_{\min}$.

5.1 INTRODUCTION

In this chapter, we consider the *online* versions of the classical scheduling problems on uniform machines $Q||C_{\max}$ and $Q||C_{\min}$ (cf. [Section 1.1](#)). In online scheduling, jobs are inserted or deleted over time, and the assignment of jobs to machines must incorporate these changes in the job set immediately. The run time of algorithms for online problems is usually measured as the time it takes to process one step, e. g. add or remove a single job. In the classical online setting, as first considered by Graham [\[58\]](#), as soon as a job is added to or removed from the instance, one must assign it to a machine, without changing the assignment of any other jobs. As Sanders, Sivadasan and Skutella [\[112\]](#) noted, it is impossible to maintain an optimal solution already for the problem with parallel identical machines $P||C_{\max}$, where all machines have the same speed $s_i = 1$, even if jobs are only inserted, unless $\mathcal{P} = \mathcal{NP}$.

For online problems, the quality of solutions is thus instead measured by the competitive ratio α of an algorithm, where an algorithm is α -competitive if it always maintains solutions within a factor α of the optimal offline solution. By this definition, α -competitive algorithms return α -approximate solutions at each step. For $Q||C_{\max}$, it is known that no better competitive ratio than 2.564 can be achieved without reassignment of jobs [\[44\]](#). If one does allow a (small) number of jobs to be reassigned at each step however, the goal now becomes to design an algorithm with small competitive ratio α , while simultaneously bounding this reassignment in some measure. The bound on the reassignment is often done in terms of the processing time of the added and removed jobs [\[10, 45, 112, 117, 121\]](#), as first introduced by Westbrook [\[121\]](#), and called *migration*.

Online scheduling problems with bounded migration on parallel identical or uniform machines appear in various applications. For example, when minimizing the makespan, the machines may represent communication channels with identical or uniformly related bandwidth, and the jobs requests for bandwidth on these channels, where the goal is to minimize the maximum load, i. e. makespan, over all channels [\[10, 121\]](#). The same problem also arises in distributed database platforms [\[10, 121\]](#), where machines represent distributed databases storing the same data, and jobs represent database accesses. The completion time of a machine then models the time required to complete all database accesses, and the aim is to minimize the max-

imum access response time. In both scenarios, jobs arrive or depart over time, and may be reassigned to any other machine for some cost that can be modeled in the migration framework [121]. An application for minimizing the maximum completion time can be found in storage area networks (SAN) design [112], where storage disks, modeled as jobs, are assigned to subservers, modeled as identical machines, and the same data is stored on each subserver to maximize fault tolerance. Adding a new disk to, or removing a faulty disk from, the system should not lead to a complete reconfiguration of the system, but instead should be reasonably expected to depend on the size of the disk, which corresponds to bounded migration in the scheduling model [112]. The generalization to uniform machines seems natural for this application, as subservers may have different capacity with respect to the total load of disk space they are able to handle efficiently.

$Q||C_{\max}$ is a generalization of the problem of scheduling jobs on parallel identical machines $P||C_{\max}$, which is already strongly $\mathcal{NP}$ -hard in the offline setting, but both machine settings admit an offline EPTAS [17, 81]. The same relationship holds also for $Q||C_{\min}$ and $P||C_{\min}$, which is also already strongly $\mathcal{NP}$ -hard in the offline setting. While there exists an EPTAS for $P||C_{\min}$ [117], for $Q||C_{\min}$, currently only a PTAS is known [13]. Formally, in the online versions of these problems, the set of jobs to be scheduled is not fixed, but instead extended or reduced in discrete steps q . At the start ($q = 0$), we are given an initial job set $\mathcal{J}_0$, and at any step thereafter ($q \geq 1$), a single job is added and its processing time revealed, or a single job is removed from the job set. We call this setting for online scheduling problems (fully) dynamic. One may also restrict the setting so that jobs may only be added, which we call permanent.

Let Δ denote the difference in value of a variable, set or function compared to itself from one specified step to another, and let $\Delta\mathcal{J}$ be the difference of the job set $\mathcal{J}$ between steps q and $q + 1$. An online algorithm has *bounded amortized migration* with amortized migration factor β , if for any step, the overall processing time ξ of jobs moved (from one machine to another) satisfies $\xi \leq \beta \left(\sum_{j \in \Delta\mathcal{J}} p_j + \Delta\Phi \right)$ for some non-negative potential function Φ that is 0 at the beginning. If $|\Delta\Phi|$ is upper bounded by some value β' , we say that the algorithm has bounded reassignment potential β' . This amortized notion of the migration factor is e. g. used by Skutella and Verschae [117]. If the reassignment potential is not bounded, this can lead to large changes at a single step, as e. g. is the case for the PTAS for bin packing in the permanent setting [74], where sometimes the whole instance has to be repacked. If $\beta' = 0$, then the algorithm has *bounded migration* with migration factor β , as used e. g. by Sanders, Sivadasan and Skutella [112]. This restricted definition implies that $\xi \leq \beta \sum_{j \in \Delta\mathcal{J}} p_j$ for any step.

5.1.1 Related Work

The first to consider approximation schemes in the migration framework were Sanders, Sivadasan and Skutella [112], who gave an $(1 + \varepsilon)$ -competitive algorithm for permanent $P||C_{\max}$, for all $\varepsilon > 0$, with migration factor $\beta \in 2^{\mathcal{O}(\varepsilon^{-1}(\log \varepsilon^{-1})^2)}$ and run time $2^{2^{\mathcal{O}(\varepsilon^{-1}(\log \varepsilon^{-1})^2)}} + \mathcal{O}(n)$, with the run time later being improved to $2^{\mathcal{O}(\varepsilon^{-2}(\log \varepsilon^{-1})^3)} + \mathcal{O}(n)$ by Ehlers and Jansen [45]. Due to the $(1 + \varepsilon)$ -competitiveness, this algorithm constitutes an online EPTAS with migration factor β . Aiming to bound the migration factor of an online PTAS in $f(\varepsilon)$ for some computable function f seems natural, as then for any fixed ε , not only is the run time polynomial, but the migration factor is constant. To this end, Sanders, Sivadasan and Skutella [112] introduce the term of *robustness*. An online scheduling algorithm is called robust if it has a constant (amortized) migration factor for any fixed $\varepsilon > 0$. Skutella and Verschae [117] managed to expand this result to both dynamic $P||C_{\max}$ and dynamic $P||C_{\min}$ by allowing some amortization with bounded reassignment potential. After showing that there exist no PTASs for general instances of dynamic $P||C_{\min}$ with constant (non-amortized) migration factor, the authors give a robust EPTAS for dynamic $P||C_{\max}$ and dynamic $P||C_{\min}$ with implicitly given amortized migration factor $\beta \in 2^{\mathcal{O}(\varepsilon^{-3}(\log \varepsilon^{-1})^3)}$, reassignment potential β' bounded by an ε fraction of the optimal objective value, and run time $2^{\mathcal{O}(\varepsilon^{-3}(\log \varepsilon^{-1})^3)} + \mathcal{O}(n)$. Note that our definition of amortized migration factor slightly differs from theirs, but is compatible.¹

Similar results also exist for related problems, such as Bin Packing in the permanent [49, 74] as well as the dynamic setting [19, 22, 51], and dynamic Knapsack [21], with newest results for both dynamic Bin Packing [22] and dynamic Knapsack [21] achieving both fully polynomial run times and migration factors. In the world of scheduling, Gálvez, Soto and Verschae [54] recently gave a robust algorithm for permanent $P||C_{\min}$ with polynomial run time and migration factor, but relaxed competitive ratio of $(4/3 + \varepsilon)$. The latest migration-related result for dynamic $Q||C_{\max}$ is by Maack [100], using a strategy employed in previous results without reassignment [12, 15] together with the migration framework to achieve a $(8/3 + \varepsilon)$ -competitive robust algorithm with amortized migration factor in $\mathcal{O}(\varepsilon^{-1})$, and a $(4 + \varepsilon)$ -competitive robust algorithm with non-amortized migration factor in $\mathcal{O}(\varepsilon^{-1})$, the latter for $\varepsilon \leq 0.8$. To the best of our knowledge however, there are no known results on robust PTASs for either $Q||C_{\max}$ or $Q||C_{\min}$, neither in the permanent nor in the dynamic setting. As

¹ Skutella and Verschae [117] define the amortized migration factor β for each step q as $\sum_{t=0}^q \tilde{\xi}_t \leq \beta \sum_{t=0}^q \sum_{j \in J_t} p_j$, where $\tilde{\xi}_t$ is the total processing time of jobs migrated at step t . They then define the accumulated reassignment potential as $K_t := \max(\tilde{\xi}_t / \beta - \sum_{j \in J_t} p_j, 0)$. It is easy to see that by setting $K_t = \Phi_t - \Phi_{t-1}$, where Φ_t is the potential just after step t , our definition of amortized migration with bounded reassignment implies theirs.

mentioned before, even in the offline setting it is not known if there exists an EPTAS for $Q||C_{\min}$ at all. A way to tackle such gaps that has become popular in recent years is usage of both approximation and parameterization in the form of EPASs [5, 26, 80, 99, 102], cf. Section 1.1 for a formal definition. Other notable ways to measure reassignment in an online setting are in the number of jobs reassigned per step, as done e. g. by Foussoul, Goyal and Kumar [53] for Fully-Dynamic Load Balancing in a restricted machine environment and by Gupta et al. [60] for Dynamic Set Cover, fixed movement costs per item, as e. g. used by Feldkord et al. [51] for Fully-Dynamic Bin Packing, or job-individual assignment costs, as used by Berndt, Eberle and Megow [20].

5.1.2 Our Contribution

Let $p_{\max}$ be the maximum processing time of any job occurring in a scheduling instance. We assume to know $p_{\max}$ a priori, a common assumption in online scheduling [28], which can also be viewed as a particular kind of advice [24, 42, 110]. This is also natural for the applications regarding communication channels, distributed database platforms and storage area networks, where $p_{\max}$ relates to the maximum allowed bandwidth per request, the maximum time required per access of a database and the maximum size of a single disk, respectively, all of which can be reasonably assumed to be upper bounded even in an online setting. As our main contribution, we achieve a robust EPTAS with amortized migration factor and bounded reassignment potential, but an additional load error of $\mathcal{O}(\epsilon p_{\max})$ on the solution, for dynamic $Q||C_{\max}$ and dynamic $Q||C_{\min}$.

An additional load error on a solution means that the loads, i. e. the total processing time of jobs on a machine, of the solution deviate by at most this error from a $(1 + \mathcal{O}(\epsilon))$ -approximate (for $C_{\max}$) or $(1 - \mathcal{O}(\epsilon))$ -approximate (for $C_{\min}$) solution. The algorithm can thus also be considered an *additive approximation scheme* as defined by Buchem et al. [25] with parameter $h = \max(\text{OPT}^*, p_{\max})$ for the optimal objective value OPT^* .

Theorem 15. *There exist robust EPTASs for both dynamic $Q||C_{\max}$ and dynamic $Q||C_{\min}$ with additional load error $\mathcal{O}(\epsilon p_{\max})$, run time $2^{\mathcal{O}(\epsilon^{-1} \log(\epsilon^{-1}) \log \log(\epsilon^{-1}))} + \mathcal{O}(|I|)$, amortized migration factor $2^{\mathcal{O}(\epsilon^{-1} \log(\epsilon^{-1}) \log \log(\epsilon^{-1}))}$, and reassignment potential bounded by $3\epsilon p_{\max}$.*

For specific instances (instances with no jobs $< \epsilon p_{\max}$), the result is a true robust EPTAS with constant migration factor.

Theorem 16. *For instances of both dynamic $Q||C_{\max}$ and dynamic $Q||C_{\min}$ without jobs of processing time less than $\epsilon p_{\max}$, there exist robust EPTASs with run time $2^{\mathcal{O}(\epsilon^{-1} \log(\epsilon^{-1}) \log \log(\epsilon^{-1}))} + \mathcal{O}(|I|)$, and migration factor $2^{\mathcal{O}(\epsilon^{-1} \log(\epsilon^{-1}) \log \log(\epsilon^{-1}))}$.*

While no true robust EPTASs for general instances, to the best of our knowledge, these results are the first in the direction of robust approximation schemes for both problems on uniform machines, as well as the first result in the direction of an efficient approximation scheme for offline $Q||C_{\min}$. This addresses an open question posed by Verschae [120] on the existence of algorithms with constant migration or reassignment factor for more general machine settings than identical machines, citing uniformly related machines as a natural generalization, as well as addresses an open question posed by Maack [100] on the existence of algorithms with competitive ratio better than 2 for scheduling on uniformly related machines in the migration framework.

It is also important to note that the need for amortization and the additional load error of Theorem 15 on the load are purely due to the small jobs with processing time $< \epsilon p_{\max}$. The use of amortization is in fact necessary here, as Skutella and Verschae [117] have shown that already $P||C_{\min}$ does not admit a robust PTAS with constant, non-amortized migration factor. To handle the small jobs in the dynamic setting of general instances, we develop a framework on how to place and migrate these jobs in an online setting, given a robust algorithm for large jobs ($p_j \geq \epsilon p_{\max}$). This framework for small jobs is similar to the widely used offline approach by Hochbaum and Shmoys [69], and thus might be of independent interest for online scheduling.

Theorem 17. *Given a robust algorithm for $Q||C_{\max}$ (resp. $Q||C_{\min}$) with run time g for jobs of size $\geq \epsilon p_{\max}$, where migration is bounded by a function $\beta(p_J)$ for each job J , there exists a robust algorithm for $Q||C_{\max}$ (resp. $Q||C_{\min}$) for all jobs with amortized migration bounded by a function $\tilde{\beta}(p_J) = 2 \cdot \beta(3p_J)$, bounded reassignment potential $3\epsilon p_{\max}$, run time at most $\mathcal{O}(g \cdot |I|)$, and with additional load error at most $\epsilon p_{\max}(4 + \lceil \frac{3}{m} \rceil) \in \mathcal{O}(\epsilon p_{\max})$.*

As an intermediate step, we also give robust EPASs for both $Q||C_{\max}$ and $Q||C_{\min}$, in both the permanent and the dynamic setting, parameterized in the maximum processing time $p_{\max}$ and the number of different processing times d occurring in the instance. We hereby extend the notion of robustness to EPASs by allowing the (amortized) migration factor to be parameterized. It suffices to a priori know upper bounds for $p_{\max}$ and d to achieve the same result (with dependencies on the bounds instead of the actual values of $p_{\max}$ and d). Note that $d \leq p_{\max}$ is a (trivial) upper bound that can always be used.

Lemma 18. *There exist robust EPASs for dynamic $Q||\gamma$ with parameters $p_{\max}$ and d , run time $2^{\mathcal{O}((d+\epsilon^{-1} \log(\epsilon^{-1} p_{\max})) \cdot \log(\epsilon^{-1} p_{\max}))} + \mathcal{O}(|I|)$, and parameterized migration factor $2^{\mathcal{O}(d \log(\epsilon^{-1} p_{\max}))}$,*

- (i) to minimize the maximum completion time (makespan), $\gamma = C_{\max}$

- (ii) to maximize the minimum completion time (machine covering), $\gamma = C_{\min}$

The general idea behind obtaining the algorithms stated in [Theorem 16](#) and [Theorem 15](#), is to first round the input values between $\varepsilon p_{\max}$ and $p_{\max}$ in such a way that $p_{\max}, d \in \mathcal{O}(\varepsilon^{-c})$ for some constant $c \in \mathbb{N}_{\geq 0}$, and afterwards, to apply the EPAS as stated in [Lemma 18](#) with the appropriate objective, directly implying these results. This rounding is done via a more sophisticated version of the classical rounding scheme [69] introduced by Berndt et al. [18], in order to get the claimed run times and migration factors.

The basic concept behind the EPASs for $Q||C_{\max}$ and $Q||C_{\min}$ is to first guess the objective value OPT^* at each step q . Then, all machines in an instance whose capacity (speed times OPT^*) is large enough to allow an additive error of $\Omega(p_{\max})$ can be covered in an ILP formulation by a single knapsack-like constraint. Their actual assignment is constructed by a simple greedy algorithm, essentially a robust version of the well-known list-scheduling by Graham [58] (see [Section 5.8](#)). Machines whose capacities are not large enough then have a capacity bounded by a polynomial in $p_{\max}$ and ε^{-1} , which allows the efficient usage of a configuration ILP based approach. The main difficulties to overcome here are the combination of both greedy and configurations based approaches for different machines in the same step, and the transitioning of machines whose capacity either becomes large enough, or too small, to be handled by the knapsack-like constraint through the addition/removal of jobs from the instance. This is the main technical contribution.

ORGANIZATION OF THIS CHAPTER We start by giving some preliminary notions in [Section 5.2](#), stating basic notation and core concepts for this chapter. We proceed in [Section 5.3](#) by giving a technical overview of our results, specifically giving some informal insight on how to obtain the EPAS for $Q||C_{\max}$, how to extend it to $Q||C_{\min}$, and how to get rid of the parameterization for both problems. We continue with all technical details, first describing how to properly manipulate the transition from a step q to $q + 1$ in order to be able to design a parameterized robust algorithm for $Q||C_{\max}$ in [Section 5.4](#), and then for $Q||C_{\min}$ in [Section 5.5](#). We show how to properly preprocess and round instances without small jobs of either problem to do the same, but without need for parameterization, in [Section 5.6](#). In [Section 5.7](#), we introduce our dynamic grouping technique to later be able to extend this result to general instances at the cost of an additional error. Afterwards, in [Section 5.8](#) and [Section 5.9](#), we give algorithms that solve one part of the partitioned problems at each step, respectively, for each of the previous settings in a robust way. We are then able to combine all our previous technical results into the robust EPASs and EPTASs for both $Q||C_{\max}$ and $Q||C_{\min}$ in [Section 5.10](#), proving our main

results. We give a brief justification for our use of high-multiplicity encoding in [Section 5.11](#), and finish this chapter with a short conclusion and an outlook on future research in [Section 5.12](#).

5.2 PRELIMINARIES

NOTATION We use migration, (amortized) migration factor and reassignment factor as defined in the introduction, where the (amortized) migration factor is called “parameterized” if β is dependent on some parameter k . We now are able to define robustness for both EP-TASs and EPASs. For optimization problems, a robust EPTAS is an online algorithm that is an EPTAS at every step q , with (amortized) migration factor dependent only on ε . Similarly, for optimization problems, a robust EPAS is an online algorithm that is an EPTAS at every step q , with (amortized) migration factor dependent only on ε and k .

If not stated otherwise, from now on we exclusively deal with high-multiplicity instances and schedules in $HM(n)$ encoding as defined in [Section 1.1](#). Recall that, for this definition, jobs of the same type have become indistinguishable. Adding a job to, or removing it from, the job set $\mathcal{J}$ is akin to adding/removing a job of this type to/from the multiplicity vector ν . We justify the use of this encoding in [Section 5.11](#). We use the operator $\nabla^{(q)}$ to denote a variable, set or function at a specific step q . Usually when Δ is used for two different steps q and q' , these steps are clear from context. Depending on the context, the minimal makespan/maximal minimum completion time over all schedules for some multiplicity vector $\nabla^{(q)}\nu$ at step q is called $\text{OPT}^*(\nabla^{(q)}\nu)$, $\nabla^{(q)}\text{OPT}^*$ if the job set is evident, or simply OPT^* if both job set and step are evident. The goal of dynamic $Q||C_{\max}/Q||C_{\min}$ is then to find a schedule with minimal makespan/maximal minimum completion time for every step q over the duration, where either $\nabla^{(q+1)}\nu = \nabla^{(q)}\nu + e^{(j)}$, for some $j \in [d]$, or $\nabla^{(q+1)}\nu = \nabla^{(q)}\nu - e^{(j)}$, for some $j \in [d]$ and $\nabla^{(q)}\nu_j > 0$.

5.2.1 Sensitivity of ILPs

Robust algorithms often make use of *sensitivity* of a subproblem. Formally, sensitivity measures how much a solution changes in relation to a change of the input. A very strong tool that is often applied in the design of such algorithms (as in the case of the aforementioned robust algorithms for $P||C_{\max}$) is the following classical result by Cook et al. [\[36\]](#) for general ILPs.

Lemma 19 (Cook et al. [\[36\]](#), Theorem 5). *Let $A \in \mathbb{Z}^{M \times N}$ and $c \in \mathbb{Z}^N$. Given an optimal integer solution x^* to the ILP $\min c^\top x$ subject to $Ax = b$,*

$x \geq \mathbb{0}$, if the modified ILP $\min c^\top x$ subject to $Ax = b'$, $x \geq \mathbb{0}$ is feasible, then there is an optimal solution $\tilde{x}$ to the latter such that

$$\|\tilde{x} - x^*\|_1 \leq \|b - b'\|_1 N\delta ,$$

where δ the largest absolute value of any subdeterminant of A .

It is well-known that, via slack variables, one can also allow inequalities at the cost of increasing only the number of variables and constraints by a constant factor. Such bounds are generally referred to as *sensitivity bounds* and are characterized by the factor on the RHS, which in case of the above result is $N\delta$. However, for uniform machines, as far as we are concerned, there is no ILP formulation known where the largest absolute value of any subdeterminant and the change of the RHS is reasonably bounded in order to accommodate insertion/deletion of a job. The general simple estimation for the largest absolute value of any subdeterminant would be $\|A\|_\infty^{\min\{M,N\}}$ (which can be strengthened a little using the Hadamard inequality, but this leads to the very same problems).

If we were to apply the Cook bound to known formulations, i. e. the [Configuration-ILP](#) or [Assignment-ILP](#), after some standard rounding procedure depending on T , the result is always a sensitivity bound that is dependent not only on ε^{-1} – which is what we would need if we were to use the resulting bound to design a robust algorithm. A different way to deal with such problems is to utilize special structural properties of ILPs that admit smaller bounds with respect to the run time and sensitivity in the ILP's parameters. One such kind of structural property is N -fold block structure (cf. [Section 1.1](#)). Such ILPs are particularly interesting in our case, as they admit a small sensitivity bound in the block parameters.

Lemma 20 (Jansen, Lassota and Rohwedder [79], Theorem 16). *Let x be an optimal solution of an N -fold ILP with RHS b , in particular, $Ax = b$. If the RHS changes to b' and the N -fold ILP still has a finite optimum, then there exists an optimal solution x' for b' ($Ax' = b'$) with*

$$\|x - x'\|_1 \leq \mathcal{O}(RS\|A\|_\infty)^{RS} \cdot \|b - b'\|_1 .$$

5.2.2 Lexicographically Minimal Vectors

We make use of a lexicographic order on (partial) solution vectors, which was also used in the prior work of Skutella and Verschae [117]. We write $a \preceq^{(\text{lex})} b$ if a is lexicographically smaller than b . It is well-known that $\preceq^{(\text{lex})}$ is a total order, and thus for any finite, non-empty set of vectors M there is a unique minimal element in M .

Now, let us consider $n \in \mathbb{N}_{\geq 0}$, $\Delta \in \mathbb{N}_{\geq 0}$, and some set

$$M \subseteq \{ v \in \mathbb{N}_{\geq 0}^n \mid \|v\|_1 \leq \Delta \} .$$

Optimizing the following linear objective does result in the lexicographically minimal solution:

$$\sum_{i \in [n]} (\Delta + 1)^{n-i} x_i .$$

To prove this more formally, let x^* be the lexicographically minimal vector in M and x a vector in M that optimizes our objective. We prove the claim by contradiction, i. e. assume that $x \neq x^*$. Then there is an index j such that $x_i = x_i^*$ for all $i \in [j-1]$, but $x_j \neq x_j^*$. If $x_j^* > x_j$, then $x \prec x^*$ which contradicts the fact that x^* is lexicographically minimal. If $x_j^* < x_j$, then

$$\begin{aligned} & \sum_{i \in [n]} (\Delta + 1)^{n-i} x_i^* - \sum_{i \in [n]} (\Delta + 1)^{n-i} x_i \\ &= \sum_{i \in [j..n]} (\Delta + 1)^{n-i} x_i^* - \sum_{i \in [j..n]} (\Delta + 1)^{n-i} x_i \\ &= (\Delta + 1)^{n-j} (x_j^* - x_j) + \sum_{i \in (j..n)} (\Delta + 1)^{n-i} x_i^* - \sum_{i \in (j..n)} (\Delta + 1)^{n-i} x_i \\ &\leq -(\Delta + 1)^{n-j} + \sum_{i \in (j..n)} \Delta (\Delta + 1)^{n-i} \\ &< 0 . \end{aligned}$$

Thus, we get that x^* has a better objective value than x , which is a contradiction to x being an optimal solution.

It is easy to see that this result can be extended to any set $\mathcal{D}$ of indices with a given total order on them. For our scheduling problem, our indices will be of the form t, c , where $t \in [\tau]$ is a machine type and c is a configuration. The order $\sqsubseteq$ we use for these indices is as follows:

$$\begin{aligned} (t, c) \sqsubseteq (t', c') &\iff s_t^{-1} p^\top c > s_{t'}^{-1} p^\top c' \\ &\vee (s_t^{-1} p^\top c = s_{t'}^{-1} p^\top c' \wedge t < t') \\ &\vee (s_t^{-1} p^\top c = s_{t'}^{-1} p^\top c' \wedge t = t' \wedge c \preceq^{(\text{lex})} c') . \end{aligned}$$

This means that we primarily order the indices by the corresponding completion time descendingly, that is by $(t, c) \mapsto s_t^{-1} p^\top c$. We then break ties by preferring the machine type of smaller index, and for same types the configuration that is lexicographically smaller.

5.2.3 An Exact Approach

It is well known that the [Assignment-ILP](#), used to describe e. g. of-line $Q||C_{\max}$, is an N -fold ILP with block parameters $R = d$, $H = d$, $S = 1$, $N = n$, and $\|A\|_\infty = p_{\max}$ [90]. To use it for both dynamic $Q||C_{\max}$ and dynamic $Q||C_{\min}$, the idea is to add an objective (and auxiliary variables) to the [Assignment-ILP](#) in order to maintain solutions that are lexicographically minimal if converted to a solution of

the [Configuration-ILP](#). However, we have to adjust the definition of lexicographically minimal solutions slightly: Configurations/Assignments to a machine of type t with the same load are now assigned the same objective value. This does not change the properties of lexicographically minimal solutions used in this chapter, except for the uniqueness of solutions. In this section, we refer to this modified variant of solutions that are lexicographically minimal after being converted to a solution to the [Configuration-ILP](#) simply by *lexicographically minimal*.

The idea to incorporate this as a linear objective is simple: For each machine and each possible load, we introduce an auxiliary 0-1-variable that is 1 iff the assignment on machine i has the corresponding load. This has two problems to overcome: First, the number of variables necessary per machine could potentially be unbounded. Second, the coefficients could become arbitrarily large.

To overcome the first problem, we proceed as follows: For an instance at step q , we compute a solution where each machine i has load at most $\nabla^{(q)} \text{OPT } s_i + p_{\max}$ and at least $\nabla^{(q)} \text{OPT } s_i - p_{\max}$ using a greedy algorithm. Let $\tilde{T}$ denote its makespan. Clearly, if we add or remove a job, the possible load values we need to consider are in $[\nabla^{(q)} \text{OPT } s_i - 2p_{\max}; \nabla^{(q)} \text{OPT } s_i + 2p_{\max}]$. Therefore, let us define

$$\Gamma: [m] \times \mathbb{N}_{\geq 0} \rightarrow 2^{\mathbb{N}_{\geq 0}}, (i, T) \mapsto \{ k \in [-2p_{\max} .. 2p_{\max}] \mid s_i T + k \geq 0 \},$$

i.e. $\Gamma(i, \tilde{T})$ is the set of possible load values for machine i if $\tilde{T}$ is the makespan of the greedy schedule. Observe that

$$\left| \bigcup_{i \in [m]} \Gamma(i, \tilde{T}) \right| \leq (4p_{\max} + 1) \cdot \tau.$$

so there are only a few makespans we need to consider for our objective. We can then introduce $4p_{\max} + 1$ auxiliary variables for each machine, each representing a specific load/makespan. Let $\lambda: [m] \times \mathbb{N}_{\geq 0} \rightarrow \mathbb{N}_{\geq 0}$ be the function that maps a machine and a load to the corresponding coefficient of (the modified variant of) $\varphi^{(\text{lex})}$.

However, if we use $\lfloor Ts_i \rfloor$ as coefficient for a variable that represents makespan T on machine i , the coefficients are unbounded, as stated before. But, since there are only $4p_{\max} + 1$ different possible makespans per machine, we can reduce the coefficients in each constraint where $\lfloor Ts_i \rfloor$ occurs by putting this value on the RHS of the ILP, and with our auxiliary variable measure how much the load assigned to any machine deviates from this value. Since the deviation is at most $2p_{\max}$, this reduces the largest coefficient to this value. This approach is captured in the following ILP formulation.

$$\begin{aligned} \min \quad & \sum_{i \in [m]} \sum_{k \in \Gamma(i, \tilde{T})} \lambda(i, \lfloor s_i \tilde{T} \rfloor + k) \cdot y_{i,k} \\ \text{s.t.} \quad & \end{aligned}$$

$$\sum_{i \in [m]} x_{i,j} = v_j \quad \forall j \in [d]$$

$$\sum_{j \in [d]} p_j x_{i,j} \leq s_i \tilde{T} + \sum_{k \in \Gamma(i, \tilde{T})} y_{i,k} k \quad \forall i \in [m]$$

$$\sum_{k \in \Gamma(i, \tilde{T})} y_{i,k} = 1 \quad \forall i \in [m]$$

with variables

$$\text{var } y_{i,k} \in \mathbb{N}_{\geq 0} \quad \forall i \in [m], k \in \Gamma(i, \tilde{T})$$

$$\text{var } x_{i,j} \in \mathbb{N}_{\geq 0} \quad \forall (i, j) \in [m] \times [d]$$

with parameters

$$\text{param } \tilde{T} \in \mathbb{Q}_{\geq 0}$$

Note that for $\tilde{T} = \nabla^{(q)} \text{OPT}$, this ILP returns an optimal schedule for both q and $q + 1$, by setting $v = \nabla^{(q)} v$ and $v = \nabla^{(q+1)} v$ respectively. Since the computed solutions are lexicographically minimal, and we allow loads up to a $p_{\max}$ deviation from $\nabla^{(q)} \text{OPT}$ in the ILP, the solution x^* computed at step $q + 1$ is also an optimal solution for $\tilde{T} = \nabla^{(q+1)} \text{OPT}$. Thus, dynamic $\text{Q}||C_{\max}$ can be solved by guessing OPT at each step q , and then solving the ILP for both steps q and $q + 1$ with $\tilde{T} = \nabla^{(q)} \text{OPT}$ and the respective job sets v .

Since we only added $\mathcal{O}(p_{\max})$ auxiliary variables and one constraint per machine compared to the [Assignment-ILP](#), the proposed ILP still is an N -fold ILP, with block parameters $R = d, H = d, S = 1, N = n$, and $\|A\|_{\infty} = p_{\max}$. The ILP can thus be solved at any step q in time $(p_{\max})^{\mathcal{O}(d^2)} \mathcal{O}(n)$ using [Lemma 1](#). Since, for computing solutions at steps q and $q + 1$, the only change in the RHS is the change in the job set, $\|b - b'\|_1 = 1$ holds for RHS b at q , b' at $q + 1$. Therefore, the sensitivity, and by extension the migration factor, is bounded by $(p_{\max})^{\mathcal{O}(d)}$ using [Lemma 20](#). This result can easily applied to solve dynamic $\text{Q}||C_{\min}$ as well, by simply changing “ $\leq$ ” to “ $\geq$ ” in the second constraint. The rest then follows equivalently. We thank an anonymous reviewer for pointing out this approach and the necessary modifications to us. However, directly applying a standard rounding scheme (e. g. [\[9, 65\]](#)) on this approach in order to obtain an EPTAS, as was successfully done before for the offline problems, suffers from the same problems as before, even with the special sensitivity: The sensitivity bound is dependent not only on ε^{-1} , and the resulting algorithm therefore not robust.

5.3 TECHNICAL OVERVIEW

For readability purposes, in this section, we give a technical overview of our results. Our results stem from a new N -fold ILP formulation for the considered problems with additionally overall small dimension and coefficients, and a non-standard rounding scheme depending

on $p_{\max}$ instead of T . This allows us to use the sensitivity result by Jansen, Lassota and Rohwedder [79] to accommodate changes of the RHS by a bounded change of the solution. We combine this with a carefully designed technique to bound the changes of the RHS in terms of the size of the inserted/removed job to obtain bounded migration. In total, bounding the change of the RHS in $\text{poly}(p_{\max}, \varepsilon^{-1})$ in the parameterized setting, and in $\text{poly}(\varepsilon^{-1})$ for the non-parameterized setting is enough to get the claimed migration factors and run times for the EPASs and EPTASs, respectively. Moreover, the small overall dimensions allow us to use a state-of-the-art algorithm for general ILPs, which yields better run time in the parameters in the approximate parameterized setting (Lemma 18) compared to the exact approach, as well as an overall smaller run time in the non-parameterized, purely approximate setting.

5.3.1 Construction of the Parameterized Robust Algorithm

In this section, we aim to give a high-level idea on the proof of Lemma 18, (i), i.e. the EPAS for dynamic $Q||C_{\max}$. We achieve our results by constructing an algorithm and proving all the claimed properties for it. This parameterized robust algorithm can then be applied to an ILP formulation of $Q||C_{\max}$ at step q , transforming the ILP to a formulation for $Q||C_{\max}$ at step $q + 1$ while keeping the change of the RHS between the ILP at step q and the ILP at step $q + 1$ bounded in $\text{poly}(p_{\max}, \varepsilon^{-1})$. All technical details of the proof for this algorithm can be found in Section 5.4. This then allows us to use Lemma 20 to bound the change in solutions, and thus the migration, between q and $q + 1$ in terms of $p_{\max}$ and ε^{-1} . Afterwards, we give an overview on how to modify these ideas to achieve the EPAS for dynamic $Q||C_{\min}$, i.e. how to prove Lemma 18, (ii), and how to get rid of parameterization altogether, i.e. how to obtain Theorem 15 and Theorem 16, as well.

By definition of OPT^* , any solution where the load of any machine i is smaller or equal to its capacity s_i OPT^* is an optimal solution for $Q||C_{\max}$. The construction of the parameterized robust algorithm is subdivided as follows: First, we show how to obtain such an algorithm if we are ensured that each machine exceeds a given minimum capacity. Then, we show how to do this if all machines subceed this very threshold. Eventually we show how we can combine these approaches to get rid of assumptions about machine capacities, which is by far the most difficult part. The upcoming (sub-)sections make use of the threshold

$$\ell := (1 + \varepsilon)^{1 + \lceil \log_{1+\varepsilon}(2\varepsilon^{-1}p_{\max}) \rceil} .$$

Note that

$$\varepsilon(1 + \varepsilon)^{-1}\ell \geq \varepsilon\ell \geq 2p_{\max} , \quad \text{and } \ell \leq \text{poly}(\varepsilon^{-1}, p_{\max}) .$$

In the following, assume that machine speeds are all powers of $(1 + \varepsilon)$. We can achieve this property by rounding each machine speed up to the next power of $(1 + \varepsilon)$. This process introduces an additional multiplicative error of at most $(1 + \varepsilon)$ on the solution. Let τ denote the number of different machine speeds after rounding. Let $s \in \mathbb{Q}^\tau$ be the vector of rounded machine speeds and $\mu \in \mathbb{N}_{\geq 0}^\tau$ the corresponding multiplicity vector. We write $s_{\max}$ for the largest speed after rounding and $s_{\min}$ for the smallest.

CASE 1: ALL MACHINES (ALWAYS) HAVE LARGE CAPACITY Here, we first assume that at all times, every machine has a capacity of *at least* ℓ w.r.t OPT^* . To compute approximate solutions, here, we use a simple adaptation of the list-scheduling algorithm [58] that allows for insertion and deletion of jobs with migration bounded in $p_j + p_{\max}$ for insertion/deletion of a job of type j , returning a solution at every step q where each machine has completion time at most $\frac{p^\top v}{s^\top \mu} + p_{\max} \leq \text{OPT}^* + p_{\max}$. Since $\varepsilon \ell \geq p_{\max}$, and all machines of each type t have capacity $s_t \text{OPT}^* \geq \ell$ by assumption, this is a $(1 + \varepsilon)$ -approximate solution at every step q due to

$$(1 + \varepsilon)s_t \text{OPT}^* \geq s_t \text{OPT}^* + \varepsilon \ell \geq s_t \text{OPT}^* + p_{\max}.$$

Details can be found in [Section 5.8](#).

CASE 2: ALL MACHINES (ALWAYS) HAVE SMALL CAPACITY Now, we assume that at all times, every machine has a capacity of *at most* ℓ w.r.t OPT^* . Then, all we need is the fact that $\ell \leq \text{poly}(\varepsilon^{-1}, p_{\max})$, i.e. ℓ is bounded by some fixed polynomial in ε^{-1} and $p_{\max}$. To obtain a solution at step q , we can set up a [Configuration-ILP](#) with $u = T$ in the configuration set for a current guess on the makespan T , discard machine types where $s_t T < 1$, resulting in at most $1 + \log_{1+\varepsilon} \ell \in \mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1} p_{\max}))$ machine types that can be assigned a non-zero configuration. We can then solve this using a standard ILP algorithm parameterized by the number of constraints ($\leq d + 1 + \log_{1+\varepsilon} \ell$) and the largest coefficient ($\leq \ell + 1$), as all capacities are bounded by ℓ . However, doing this makes going from step q to $q + 1$ – in contrast to the case where all machines have large capacities – not that straightforward. This is only due to removal: If we were only allowed to add jobs, we could simply increase T when necessary, possibly include some newly available configurations into the ILP, convert our old solution to a solution of the new ILP by adding “0”s for the new entries (those that correspond to newly introduced configurations) and then use the result by Jansen, Lassota and Rohwedder [79] to find a solution nearby. However, if we can remove jobs, we cannot convert our old solution to one with smaller T in general, as we might have to change the assignment of $\Theta(m)$ machines, if they are assigned a configuration that has to be removed from the ILP due to the decrease of T .

This problem also occurs for identical machines, and Skutella and Verschae [117] solved it in the following way: They list a superset of necessary configurations and use a linear objective $\varphi^{(\text{lex})}$ that encodes “Minimizing the makespan”. We proceed similarly, as – due to the assumption that capacities never exceed ℓ – we can simply list all configurations whose load does not exceed ℓ . We additionally need to generalize the construction of the linear objective such that it can be applied to uniform machines with a few machine types, which is rather straightforward. Recall that at most $1 + \log_{1+\varepsilon} \ell \in \mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1} p_{\max}))$ machines can be assigned a non-zero configuration at each step, if we list only machine types that – before or after the operation – can actually be assigned a job, so this number is still $\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1} p_{\max}))$. We then achieve the desired result using the sensitivity bound as stated in Lemma 20. Note that, since the such constructed objective leads to unique optimal solutions, to compute a new solution it suffices to simply *solve* the ILP, while only implicitly bounding the changes of variables. Details on this approach, both for the sensitivity bound and the actual solving of the ILP, can be found in Section 5.9. This is also where the run time and migration factor of the algorithms come from, as solving the ILP and bounding the change in parameters is the bulk of the computational work.

COMBINING BOTH CASES Let us call the algorithm for only large capacities “blue algorithm”, and the one for only small capacities “red algorithm”. Our goal is now to partition the jobs and machines into red and blue jobs and machines at each step q , respectively, using the blue algorithm for blue jobs and blue machines, and the red algorithm for red jobs and machines to compute feasible approximate solutions for both sets in the partition, and by that, for the whole instance. Let the vector $\mu^{(\text{blue})} \in \mathbb{N}_{\geq 0}^{\tau}$ be the vector of blue machines, $\nu^{(\text{blue})} \in \mathbb{N}_{\geq 0}^d$ be the vector of blue jobs at some step q .

Consider (ILP^{dynamic}), an ILP formulation that is parameterized by ν , $\mu^{(\text{blue})}$ and some value $\alpha \in \mathbb{Q}_{\geq 0}$.² This ILP amounts to a modified version of the Configuration-ILP (with upper bound ℓ on the configurations of each machine type), extended by a knapsack-like constraint with capacity α dependent on the total capacity of the blue machines. The total capacity on the blue machines is itself dependent on the current number of blue machines $\mu^{(\text{blue})}$. The parameter ν gives us the current multiplicity of each job type in the instance. The objective function $\varphi^{(\text{lex})}$ again encodes “Minimizing the makespan” on the given set of configurations.

² A technical detail: for ILPs, the RHS is usually constrained to be integral. We round it down before applying Lemma 20 or running an ILP solver, covering the details where this occurs. For the rest of the work we use fractional values, which simplifies our statements.

(ILP^{dynamic})

$$\min \varphi^{(\text{lex})}(x)$$

s.t.

$$p^\top \nu^{(\text{blue})} \leq \alpha \quad (5.1)$$

$$\sum_{c \in \mathcal{C}(\ell)} x_{t,c} = \mu_t - \mu_t^{(\text{blue})} \quad \forall t \in [\tau] \quad (5.2)$$

$$\sum_{\substack{t \in [\tau] \\ c \in \mathcal{C}(\ell)}} x_{t,c} \cdot c + \nu^{(\text{blue})} = \nu \quad (5.3)$$

with variables

$$\text{var } x_{t,c} \in \mathbb{N}_{\geq 0} \quad \forall t \in [\tau], c \in \mathcal{C}(\ell)$$

$$\text{var } \nu^{(\text{blue})} \in \mathbb{N}_{\geq 0}^d$$

with parameters

$$\text{param } \nu \in \mathbb{N}_{\geq 0}^d$$

$$\text{param } \mu^{(\text{blue})} \in \mathbb{N}_{\geq 0}^\tau$$

$$\text{param } \alpha \in \mathbb{Q}_{\geq 0}$$

Variables $x_{t,c}$ represent the configurations of red machines, and thus encode all jobs placed on these machines. Variables $\nu^{(\text{blue})}$ represent all jobs placed on blue machines instead. [Constraint 5.1](#) ensures that the total load of jobs on blue machines represented by $\nu^{(\text{blue})}$ does not exceed the currently available capacity on blue machines α . [Constraint 5.2](#) guarantees that the number of configurations of each machine type t in any solution matches the current number of red machines $\mu_t - \mu_t^{(\text{blue})}$ of this type. Finally, [Constraint 5.3](#) makes sure that each job is either assigned via a configuration to a red machine, or assigned to the blue machines via $\nu^{(\text{blue})}$.

Let T be a guess on the optimal makespan at some step q . We are now able to compute an approximate solution for dynamic $Q||C_{\max}$ at step q by setting $\mu_t^{(\text{blue})} := \mu_t$ if $s_t T \geq \ell$ and $\mu_t^{(\text{blue})} := 0$ otherwise, for each $t \in [\tau]$, and setting $\alpha := Ts^\top \mu^{(\text{blue})}$, i.e. allowing the total capacity of the blue machines to be used. Using the red algorithm on such a modified ILP then computes a solution with makespan T , if it exists, on the red machines for some subset of the jobs $\nu - \nu^{(\text{blue})}$, such that the load of jobs not placed on the red machines $p^\top \nu^{(\text{blue})}$ is smaller or equal to the total capacity on the blue machines $Ts^\top \mu^{(\text{blue})}$. If such a solution exists, using the blue algorithm to compute an assignment of $\nu^{(\text{blue})}$ on $\mu^{(\text{blue})}$ then returns a solution with makespan at most $(1 + \varepsilon)T$. Combining these solutions gives us a schedule with a makespan of at most $(1 + \varepsilon)T$ over all machines. If no solutions exists to the ILP with makespan T , we can correctly conclude there exists no solution with this makespan.

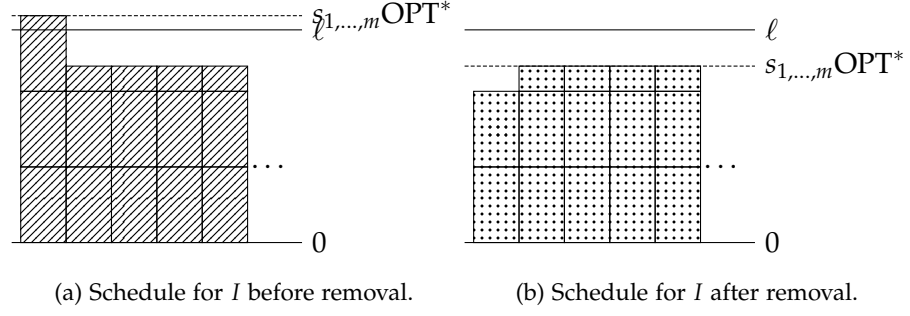


Figure 5.1: Example of potentially unbounded parameter change for an instance I .

However, to go from step q to $q + 1$, we need to repartition the set of machines into blue and red machines depending on the possibly different optimal makespan, a process we call *recolouring*. This recolouring affects $\mu^{(\text{blue})}$, changing the RHS of $(\text{ILP}^{\text{dynamic}})$, and thus impacting the bound on the sensitivity and the migration factor. In fact, setting up $(\text{ILP}^{\text{dynamic}})$ in such a way that, at each step q , using the parameterized red and blue algorithms to compute *robust* solutions turns out to be quite challenging. The two main problems are (i) that we might have to recolor $\Omega(m)$ machines, if their capacities cross the threshold ℓ due to changes in the optimal makespan; and (ii) the RHS of the knapsack constraint α is highly dependent on the optimal makespan: even a tiny change in the optimum can trigger a $\omega_{\varepsilon, p_{\max}, d}(1)$ change. Both scenarios would directly lead to unbounded migration.

To gain some more insight into this issue, consider the following example: Let I be an instance with speeds $s_1 = \dots = s_m = 1$, $2m + 1$ jobs of size $p_{\max}$, and $m - 1$ jobs of size $\frac{p_{\max}}{3}$. Let ε be chosen such that $3p_{\max} > \ell$ but $2p_{\max} + \frac{p_{\max}}{3} < \ell$. A schedule might look as pictured in Figure 5.1a: By pigeon-hole, there must be at least one machine i with at least 3 jobs of size $p_{\max}$. Thus, the capacity of all machines for makespan OPT^* must be greater than ℓ . All machines are colored blue (pictured as north-east-line patterned). The placement of the remaining jobs may be any solution that results from a greedy assignment, e. g. each machine is assigned 2 jobs of size $p_{\max}$ and one job of size $\frac{p_{\max}}{3}$. Now assume 1 job of size $p_{\max}$ is removed from i , i. e. the machine with 3 jobs of size $p_{\max}$: Then, OPT^* immediately drops to $p_{\max} + \frac{p_{\max}}{3}$. In such a case, we get a situation as pictured in Figure 5.1b: All m machines have to be recolored red (pictured as dotted) and α drops from $> m\ell$ to 0. This leads to an unbounded change in parameters, and thus unbounded migration as measured by sensitivity, even though for a lexicographically minimal solution on the newly red machines, no jobs have to be migrated.

Our approach to tackle this is twofold: (i) we discretize the makespan guarantees in order to not be forced to react to small changes all the time; and (ii) we distribute the work that needs to be done when the

makespan guarantee changes over the preceding steps, such that in the step where this change occurs, the remaining necessary changes are properly bounded. The latter means that we recolor machines over time such that when our guarantee changes, there are at most a few machines left to recolor; and analogously we change the RHS of the knapsack constraint over time. First, let us capture the restriction on the machine coloring by an invariant:

Invariant 3.A (Machine Coloring Invariant). Given a bound $T \in \mathbb{Q}_{\geq 0}$ on the objective value, for any blue machine of type t we have $s_t T \geq \ell$; and for any red machine of type t we have $s_t T \leq \ell$.

Since our ILP is parameterized by the value α that represents the part of the overall capacity on blue machines that may be used by the ILP, our next step is to formalize this interaction, i. e. ensure that we restrict ourselves to values of α that do not exceed the overall total capacity of the blue machines, for which we now allow another additional multiplicative error of $(1 + \varepsilon)$.

Invariant 3.B (Compatible Coloring Invariant).

$$0 \leq \alpha \leq (1 + \varepsilon) T s^\top \mu^{(\text{blue})} .$$

For fixed T and $\mu^{(\text{blue})}$, we will also call $(1 + \varepsilon) T s^\top \mu^{(\text{blue})}$ the “available blue area (for T and $\mu^{(\text{blue})}$)”. As mentioned before, the direct application of [Lemma 20](#) does not lead to a sufficiently small sensitivity bound, due to possible large differences in the RHS b and b' – this is the main issue to overcome. Given a (approximate) makespan T , we require the red schedule to not exceed T , while for the blue schedule we only require a makespan of $(1 + \varepsilon)T$. This will later allow us to bound the number of machine recolourings by some kind of witness argument.

One property of the objective $\varphi^{(\text{lex})}$ is that, if applied to all variables, optimal solutions are unique as long as feasible solutions can be bounded by some fixed value. This property also applies to [\(ILP^{dynamic}\)](#). Our main focus thus does not rely on finding solutions that meet the requirements on the migration, but to adjust the parameters in such a way that the unique optimal solution to these parameters does so. Thus, instead of classifying solutions, we classify triples of parameter values, called “states”, calling those “ T -valid” where the optimal solutions meets our requirements (for makespan T):

Definition 10 (States, associated solutions, ILP-makespan, T -validity). A triple of the ILP parameters $(\alpha, \mu^{(\text{blue})}, \nu)$ is called *state*. The (unique) optimal solution $(x, \nu^{(\text{blue})})$ of [\(ILP^{dynamic}\)](#) for these parameters is called *the associated solution*. The *(ILP) makespan of a state* is the (ILP) makespan of the associated solution, i. e. $C_{\max}(x)$. If the ILP is infeasible, we define the makespan to be ∞ .

A state S is T -valid iff [Invariant 3.A](#) and [Invariant 3.B](#) both hold for T ; and the associated solution of S has (ILP) makespan at most T .

Note that if there is a T -valid state, there is also T' -valid state for any $T' \geq T$. Solving [\(ILP^{dynamic}\)](#) for a T -valid state directly yields a solution with makespan at most $(1 + \varepsilon)^2 T$, with the error being introduced by the additional εT error on the blue area, and the actual placement of blue jobs on blue machines (see [Section 5.8](#)). As we want to solve the ILP for (approximately) the optimal makespan OPT^* , we introduce the following definition.

Definition 11 (Optimal ILP Makespan). Given a job vector $\nu \in \mathbb{N}_{\geq 0}^d$, $\text{OPT} \in \mathbb{Q}_{\geq 0}$ is the smallest number such that there is a OPT -valid state S . Moreover, we define $\text{OPT}^\downarrow$ to be the largest power of $(1 + \varepsilon)$ strictly smaller than OPT ; and $\text{OPT}^\uparrow := (1 + \varepsilon) \text{OPT}^\downarrow$.

Definition 12 (Valid States). A state S is valid if it is $\text{OPT}^\uparrow$ -valid.

Note that there is not necessarily just one, but often times many valid states. We will exploit this heavily, by only converting a valid state to another valid state with “nice” properties, i. e. bounded migration. By the definition of OPT , $\text{OPT}^* \geq \text{OPT}$. Thus, computing a solution to the ILP for any valid state always returns an at most $(1 + \varepsilon)^3 \in (1 + \mathcal{O}(\varepsilon))$ -approximate solution. Now, we can discuss the general idea of achieving bounded migration using this framework of [\(ILP^{dynamic}\)](#), states and associated solutions. Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ be a valid state and $\bar{\nu}$ a changed job set. We assume wlog. that $\bar{\nu}$ differs from ν by exactly one job. Our goal is to always find a valid state $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ such that $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 \leq \text{poly}(p_{\max}, \varepsilon^{-1})$. As we show in [Section 5.9 \(Lemma 46 \(i\)\)](#), this directly ensures that the unique associated solutions have bounded ℓ_1 -distance and thus we obtain bounded migration. Since we require $\bar{S}$ to be valid, the associated solution to this state is then again a $(1 + \mathcal{O}(\varepsilon))$ -approximate solution to the new job set, and the algorithm therefore is a robust EPAS. We construct such an $\bar{S}$ by carefully recoloring machines and changing α from S to $\bar{S}$. To do this, we only consider machines of capacity exactly ℓ for recoloring in each step: Due to the discretization of the makespan to $\text{OPT}^\uparrow$, these are the only types of machines that are “critical”, i. e. they may be forced to be either red (if $\text{OPT}^\uparrow$ decreases) or blue (if $\text{OPT}^\uparrow$ increases) at the next makespan change, but may not be appropriately colored yet before this change. At the same time, these critical machines are the only ones that are allowed to be colored either red or blue for $\text{OPT}^\uparrow$ by [Invariant 3.A](#).

First, let us consider the case where $\Delta \text{OPT}^\uparrow = 0$, i. e. the makespan guarantee does not change. When some job of type j is added, we only increase the value of α if necessary – thus, we act some kind of “lazy”. If an increase is necessary but we cannot increase α due to [Invariant 3.B](#), we can conclude that there is some red machine

that can be recolored blue – otherwise $\text{OPT}^\uparrow$ would have increased. By doing so, due to the fact that the makespan guarantee on blue machines is relaxed to $(1 + \varepsilon) \text{OPT}^\uparrow$, we gain an additional space of at least $\varepsilon \ell \geq p_{\max}$ that can be used for the necessary increase of α . Thus, recoloring a single machine suffices in this case. When some job of type j is removed, as long as there is a blue machine that must not be blue after $\text{OPT}^\uparrow$ decreases, we recolor a single machine if possible without ending up in an invalid state, to prepare for an eventual decrease of $\text{OPT}^\uparrow$. If there is no such machine, we decrease α by $\mathcal{O}(p_{\max})$ if possible (and the maximal possible value otherwise), while ensuring that α does not subceed a specific threshold. The latter is necessary to ensure validity once $\text{OPT}^\uparrow$ has dropped, i.e. just after a drop we want α to be the maximal possible value that it may take under the new $\text{OPT}^\uparrow$ -value and its implied blue area.

An increase of $\text{OPT}^\uparrow$ only occurs when a job is added and no more critical machines can be recolored blue. Since an increase of $\text{OPT}^\uparrow$ increases the available blue area on every blue machine by at least $\varepsilon \text{OPT}^\uparrow \geq \varepsilon \ell$, we can simply increase α by as much. A decrease of $\text{OPT}^\uparrow$ implies the removal of a job, and we recolor the machines that we need to recolor (some of those that are currently blue but may not be allowed to be blue under the new value of $\text{OPT}^\uparrow$), and reduce α to ensure that it does not exceed its maximal value in respect to the new, reduced $\text{OPT}^\uparrow$ -value. Due to our continuous blue $\rightarrow$ red recoloring at each job removal, we can show that this affects only a small number of machines and a small change in α at the point of $\text{OPT}^\uparrow$ decrease, as otherwise, they could have (and would have) been recolored before, or were never colored blue for this $\text{OPT}^\uparrow$ -value in the first place. Crucial for proving this is the fact that, by the continuous recoloring, α always uses almost all available blue area, implying that if the recoloring and α changes were too big, already the job set previous to the removal would have been valid for a decreased $\text{OPT}^\uparrow$ -value – a contradiction to the validity of this state. We thus show that we go from valid state to valid state when adding or removing jobs, while changing the state parameters only in terms of $\text{poly}(\varepsilon^{-1}, p_{\max})$. As the parameters are the only changes in the RHS of $(\text{ILP}^{\text{dynamic}})$, besides the job set being changed by 1 for the inserted or removed job, this then directly implies a bound $\text{poly}(\varepsilon^{-1}, p_{\max})$ on the change of the RHS.

With this result in hand, and the fact that $(\text{ILP}^{\text{dynamic}})$ has N -fold structure, we can use the sensitivity result stated in [Lemma 20](#) to get our bound on the sensitivity, and thus, migration factor. As $(\text{ILP}^{\text{dynamic}})$ has only a relatively small number of constraints in total, the most efficient way to compute the (unique) solutions at each step q is via an algorithm for solving ILPs with small number of constraints by Jansen and Rohwedder [\[81\]](#). Details on the sensitivity and ILP solver can be found in [Section 5.9](#).

5.3.2 *Extension to Robust $Q||C_{\min}$*

The construction of the parameterized robust algorithm necessary to prove Lemma 18, (ii), i.e. handling dynamic $Q||C_{\min}$, is almost symmetrical to the parameterized robust algorithm for $Q||C_{\max}$. The high-level idea is in fact the same: We split the machine set into blue and red machines depending on their capacity at some step q w.r.t. a slightly larger fixed threshold ℓ , using this split to cover large machines ($\geq \ell$) with a single knapsack-like constraint in an ILP formulation. The blue machines keep their $\varepsilon\ell$ -increased capacity. This may at first sound counter-intuitive, as for maximizing the minimum completion time, this means we now have to cover more capacity than strictly necessary when coloring a machine blue. We can offset this by the allowed error however, and it will help us later to keep the changes bounded while coloring blue $\rightarrow$ red. We formalize this approach in a modified version of (ILP^{dynamic}). The main difference here is that, as we want to approximate the minimum completion time objective of a solution, we require the total processing time of jobs on these blue machines to be *above* some parameter α that depends on the total capacity of the blue machines. Thus, the knapsack constraint in (ILP^{dynamic}) is changed to the dual knapsack constraint $p^\top \nu^{(\text{blue})} \geq \alpha$. We also use a modified $\varphi^{(\text{lex})}$ function that maximizes the minimum completion time on the red machines by computing a lexicographically minimal solution to the dual order, i.e. to the reverse order of configurations that we used for $C_{\max}$. To make sure that the minimum completion time is large enough on the blue machines as well, we need to redefine the “compatible coloring invariant” to $\alpha \geq (1 + \varepsilon)Ts^\top \mu^{(\text{blue})}$. A solution to this modified (ILP^{dynamic}) with makespan OPT^* in the coloring is now an $(1 - 2\varepsilon)$ -approximate solution to $Q||C_{\min}$ for the job set at step q .

The main challenge, again, is now bounding the change in the RHS $\|b - b'\|_1$ to keep sensitivity, and thus migration, small. Like before, this is done via a discretization of the objective value, slow change of α , and careful recoloring of critical machines in each step. Here however, we restrict our ILP to $\text{OPT}^\downarrow$ - instead of $\text{OPT}^\uparrow$ -values, in order to get a proper approximation for the new objective. The careful recoloring and changing of α is done through similar procedures as before, taking a state of the three parameters α , $\mu^{(\text{blue})}$ and ν , and modifying them in such a way that they do not change to much, but are valid for the new step $q + 1$, i.e. the corresponding ILP does admit a solution with the proper approximate minimum completion time. The rest of the proof then follows, but “dual”, as for dynamic $Q||C_{\min}$, the difficulty of these changes is switched. As mentioned before, we now actually have to cover more capacity when coloring a machine blue. This is necessary to make the removal of jobs rather easy, as recoloring a blue machine red always frees enough load to “fill the gap” left by the removal of a

job on any machine. If we couldn't guarantee this property, removing a single job could trigger a decrease of $\text{OPT}^\downarrow$ while a number of critical machines are still colored blue, and thus an unbounded blue $\rightarrow$ red recoloring of machines. With this property however, at a decrease of $\text{OPT}^\downarrow$, we can guarantee that all critical machines are already correctly colored red and α does not decrease too much. Instead, now inserting a job requires continuous recoloring of machines red $\rightarrow$ blue, as to make sure all critical machines are recolored blue before an increase of $\text{OPT}^\downarrow$ is triggered.

5.3.3 Getting Rid of the Parameterization

With the EPASs in hand, we can now informally describe the necessary modifications made to any instance in order to achieve the EPTAS results as stated in [Theorem 16](#) and [Theorem 15](#). For this, we generally use the standard approach of reducing the complexity of an instance by preprocessing, i. e. rounding and simplifying, the input at the cost of an additional error on the solutions. In contrast to the usual approaches however, which do the rounding dependent on a guess of the optimal objective value OPT^* , our rounding depends on $p_{\max}$ instead, as OPT^* itself is not fixed but dependent on the current job set at a specific step.

Our preprocessing follows a recent improvement over the well known rounding and scaling scheme for scheduling problems. This improvement, as first described by Berndt et al. [18], aims to reduce the largest absolute entry in the configurations necessary to describe a scheduling problem from (ε^{-1}) to $\log(\varepsilon^{-1})$ after rounding. They successfully apply this technique to reduce the ℓ_1 -norm of the columns, and thus the hereditary discrepancy of the matrix, from ε^{-1} to $\log(\varepsilon^{-1})$ in a [Configuration-ILP](#) formulation for offline $P||C_{\max}$. Solving this ILP with the algorithm by Jansen and Rohwedder [81], whose run time depends on the hereditary discrepancy, then yields an EPTAS with run time $2^{\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1}))}$. As [\(ILP^{dynamic}\)](#) is an extension of [Configuration-ILP](#), and we use the same algorithm by Jansen and Rohwedder [81] to solve the ILP, we also make use of this improved rounding scheme to not only reduce $p_{\max}$ and d to values only dependent on ε^{-1} , but also to reduce the ℓ_1 -norm of the configuration part of the ILP. Additionally however, we have to reduce the largest coefficient in the other constraints of [\(ILP^{dynamic}\)](#) to $\log(\varepsilon^{-1})$ as well, in order to get the proper bound on the ℓ_1 -norm of the columns. Luckily, after applying the rounding above, the only other time where a coefficient larger than $\log(\varepsilon^{-1})$ appears in [\(ILP^{dynamic}\)](#) is in the (single) knapsack constraint. To deal with this, we use a technique introduced by Knop, Pilipczuk and Wrochna [91] called “Coefficient Reduction” to reduce the largest coefficient of this constraint to 1, while introducing an additional $\log(\varepsilon^{-1})$ constraints and variables as a trade-off. This

such rounded and reduced ILP is called $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$. Details on this preprocessing can be found in [Section 5.6](#). As our rounding scheme is dependent on $p_{\max}$, we first consider only instances with processing time values in $[\varepsilon p_{\max}, p_{\max}]$. Instances that include processing times $< \varepsilon p_{\max}$ are dealt with using our dynamic grouping technique as stated in [Theorem 17](#).

Any solution to $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$ for $Q||C_{\max}$ or $Q||C_{\min}$ can be transformed to a $(1 + \mathcal{O}(\varepsilon))$ -approximation or $(1 - \mathcal{O}(\varepsilon))$ -approximation of the original instance, respectively, and all relevant parameters for solving the ILP are now bound in terms of ε^{-1} only. Again, using the same procedures for inserting and removing jobs as in the parameterized case, we transition from step q to $q + 1$. Here, this keeps the change of the RHS bounded in terms of $\text{poly}(\varepsilon^{-1})$. Thus, by again using [Lemma 20](#) to bound sensitivity and the algorithm by Jansen and Rohwedder [\[81\]](#) to solve the ILP, we achieve a run time of $2^{\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1}))} + \mathcal{O}(|I|)$, matching the currently fastest EPTAS for offline scheduling of jobs on identical machines [\[18\]](#), and a migration factor of $2^{\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1}))}$. Details on this can be found in [Section 5.9](#), [Lemma 47](#), (ii) and [Lemma 46](#) (ii), respectively. Note that by modifying a standard rounding scheme (e.g. [\[9, 65\]](#)) to be dependent on $p_{\max}$ instead of T , results with the same approximation guarantees can also be achieved by applying this rounding to the exact approach described in [Section 5.2.3](#). However, this results in an algorithm with a worse run time of $2^{\mathcal{O}(\varepsilon^{-2} \log^3(\varepsilon^{-1}))} + \mathcal{O}(|I|)$, and a worse migration factor of $2^{\mathcal{O}(\varepsilon^{-1} \log^2(\varepsilon^{-1}))}$.

5.4 TECHNICAL DETAILS FOR ROBUST $Q||C_{\max}$

In this section, we give the full technical details necessary to prove that we can set up an ILP with parameters α , $\mu^{(\text{blue})}$ and ν at each step q that has an optimal solution $(x, \nu^{(\text{blue})})$, such that x is a $(1 + \mathcal{O}(\varepsilon))$ -competitive solution for the red machines, $\nu^{(\text{blue})}$ has $p^\top \nu^{(\text{blue})} \leq (1 + \varepsilon) \text{OPT}^\uparrow s^\top \mu^{(\text{blue})}$ for the blue machines, and $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 + \|\Delta\nu\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max})$ between any steps q and $q + 1$. As we have shown that $(\text{ILP}^{\text{dynamic}})$, as defined in [Section 5.3.1](#), already possesses the first two properties if $\varphi^{(\text{lex})}$ encodes “Minimizing the makespan”, only this fact and the last property on the parameter change are left to be proven.

5.4.1 Some Preliminaries

Before we can formulate our procedure precisely, we need some more definitions. We begin by formally describing $\varphi^{(\text{lex})}$, the objective of $(\text{ILP}^{\text{dynamic}})$, and show that it indeed encodes “Minimizing the

makespan". For this, consider (ILP^{dynamic}) (but without objective, as we are constructing it here). Let

$$X := \left\{ x \mid \exists v^{(\text{blue})}, (x, v^{(\text{blue})}) \text{ is feasible} \right\}.$$

Since for any $x \in X$ we have $\|x\|_1 \leq m$ (we cannot choose more configurations than we have machines), we can construct a linear objective as before that models lexicographic minimization with respect to our ordering over X .

Since for any $x \in X$ there is a unique value $v^{(\text{blue})}$ such that $(x, v^{(\text{blue})})$ is feasible to (ILP^{dynamic}), we get that (ILP^{dynamic}) has a unique optimal solution. This solution has a very nice property: When only considering chosen configurations, it has minimal makespan. We prove this by contradiction. Let x^* be the lexicographically minimal solution of (ILP^{dynamic}) as defined in Section 5.2.2. Assume that there was a feasible solution x to (ILP^{dynamic}) with makespan strictly smaller than x^* , i.e. $C_{\max}(x) < C_{\max}(x^*)$. Let $(t, c) \in \text{supp}(x^*)$ be the smallest element of the support of x^* with respect to our given ordering of indices. Then for all predecessors (t', c') of (t, c) we have $x_{t', c'}^* = 0$. As we primarily ordered the indices descendingly by completion time, and $C_{\max}(x) < C_{\max}(x^*)$ by assumption, we can derive that $x_{t, c} = 0$ and $x_{t', c'} = 0$ for all predecessors (t', c') of (t, c) . This implies that x is lexicographically strictly smaller than x^* , which contradicts the minimality of x^* .

Recall that for a schedule σ , we write σ_i as the vector in $\mathbb{N}_{\geq 0}^d$ with j -th entry being $\sigma(i, j)$ for each $j \in [d]$. The completion time of a machine i in a schedule σ is $\sum_{j: \sigma_i(j)} p_j / s_i$. The completion time of machine i of type t in a schedule σ can then be described as $s_t^{-1} p^\top \sigma_i$. Let $[x]_0 := [x] \cup \{0\}$.

Now, we define the "freeness" of states, e.g. how much space as measured in the parameter α is additionally allotted in some solution, even though a smaller value would suffice for a solution with the same makespan.

Definition 13 (κ -free States). A valid state $(\alpha, \mu^{(\text{blue})}, \nu)$ is κ -free if $(\alpha - \kappa, \mu^{(\text{blue})}, \nu)$ is valid.

One can interpret a κ -free state as a state where there is a solution valid to (ILP^{dynamic}) with the knapsack constraint (5.1) having slack at least κ . The general idea behind this measure is that we only want to compute states that act in a controlled manner regarding their "freeness". By this, we can ensure that, when another job is inserted or removed in the next step, the parameters $\mu^{(\text{blue})}$ and α don't change too much.

We now intend to define a partial map that maps a makespan to the machine type $t \in [\tau]$ whose real capacity (under the given makespan) is exactly ℓ , i.e. the aforementioned machines that are "critical". The map is only partial as there might simply be no type with appropriate

capacities, since there does not have to be a machine of speed $(1 + \varepsilon)^k$ for every $k \in \mathbb{Z}$.

Definition 14 (Critical Type).

$$\text{critical-type}: (1 + \varepsilon)^{\mathbb{Z}} \rightarrow [\tau] , \quad T \mapsto \begin{cases} t & \text{if } \exists t \in [\tau]. s_t T = \ell \\ \perp & \text{otherwise} \end{cases} .$$

As a shorthand we write $\dot{t} := \text{critical-type}(\text{OPT}^\uparrow)$ when $\text{OPT}^\uparrow$ is evident from context, and $\nabla^{(q)} \dot{t} := \text{critical-type}(\nabla^{(q)} \text{OPT}^\uparrow)$. Moreover, as a convention, if we consider multiple makespans and their critical types, we will always name those types based on the notation $\dot{t}$, e.g. $\bar{\dot{t}}$, $\tilde{\dot{t}}$ et cetera.

Machines of this type are the ones that might be necessary to be recolored during a sequence of steps where $\text{OPT}^\uparrow$ does not change. To see this, we need to show that OPT changes by a factor of at most $(1 + \varepsilon)$, which implies similar bounds for $\text{OPT}^\uparrow$ and $\text{OPT}^\downarrow$. This then implies that recoloring all machines of this type is enough to maintain [Invariant 3.A](#) and feasibility of the solution. In general, there is no such bound – even for identical machines, as Skutella and Verschae [117] already noted. Thus, instead of introducing a general measure on all instances, we bound the change for the relevant cases only, where at least one machine has capacity at least ℓ either before or after a change of the job set. This suffices, since – as described above – we intend to use this bound to show we only need to recolor machines $\text{red} \rightarrow \text{blue}$ or $\text{blue} \rightarrow \text{red}$ that are of critical type. Clearly, if no machine can be colored blue (without violating [Invariant 3.A](#)), there is no need (nor possibility) to do a recoloring of any kind. To show the bound on the change, we need the following weak estimation of OPT :

Proposition 3 (A Weak Estimation Of OPT). $\text{OPT} \leq s_{\max}^{-1} p^\top v$.

Proof. Let σ a schedule where all jobs are placed onto a single machine i with $s_i = s_{\max}$. Then σ has makespan $s_{\max}^{-1} p^\top v$, so it is a feasible solution for this problem with $T \leftarrow s_{\max}^{-1} p^\top v$. Thus, we know that there is a $s_{\max}^{-1} p^\top v$ -valid state, and thus $\text{OPT} \leq s_{\max}^{-1} p^\top v$ by [Definition 11](#). $\square$

With this simple estimate, we can show the following lemma.

Lemma 21 (Bounding Makespan Changes). Consider $v \in \mathbb{N}_{\geq 0}^d$ and $j \in [d]$. Let $f(\cdot) \in \{\text{OPT}(\cdot), \text{OPT}^\downarrow(\cdot), \text{OPT}^\uparrow(\cdot)\}$. If $s_{\max} \text{OPT}^\uparrow(v + e^{(j)}) \geq \ell$, then

$$\frac{f(v + e^{(j)})}{f(v)} \leq (1 + \varepsilon) ,$$

and if $j \in \text{supp}(v)$ and $s_{\max} \text{OPT}^\uparrow(v) \geq \ell$,

$$\frac{f(v - e^{(j)})}{f(v)} \geq (1 + \varepsilon)^{-1} \geq (1 - \varepsilon) .$$

Proof. We start with the first inequality and $f(\cdot) = \text{OPT}(\cdot)$. Let $\bar{v} := v + e^{(j)}$. By assumption, we have $s_{\max} \text{OPT}^\uparrow(\bar{v}) \geq \ell$. Hence

$$\begin{aligned} (1 + \varepsilon)^{-1} s_{\max}^{-1} \ell &\leq \text{OPT}(\bar{v}) \leq \text{OPT}(v) + \Delta \text{OPT} \\ \iff (1 + \varepsilon)^{-1} s_{\max}^{-1} \ell - \Delta \text{OPT} &\leq \text{OPT}(v) . \end{aligned}$$

We can thus deduce that $\Delta \text{OPT} \leq s_{\max}^{-1} p_j$ by transforming an optimal solution, placing j onto the fastest machine, and transforming the schedule back into a $\text{OPT}(\nu) + s_{\max}^{-1} p_j$ -valid solution. Moreover, obviously $\Delta \text{OPT} \geq 0$. Now,

$$\begin{aligned}
\frac{\text{OPT}(\bar{\nu})}{\text{OPT}(\nu)} &= \frac{\text{OPT}(\nu) + \Delta \text{OPT}}{\text{OPT}(\nu)} = 1 + \frac{\Delta \text{OPT}}{\text{OPT}(\nu)} \\
&\leq 1 + \frac{\Delta \text{OPT}}{(1+\varepsilon)^{-1} s_{\max}^{-1} \ell - \Delta \text{OPT}} \leq 1 + \frac{s_{\max}^{-1} p_j}{(1+\varepsilon)^{-1} s_{\max}^{-1} \ell - s_{\max}^{-1} p_j} \\
&= 1 + \frac{p_j}{(1+\varepsilon)^{-1} \ell - p_j} \leq 1 + \frac{p_{\max}}{(1+\varepsilon)^{-1} \ell - p_{\max}} \\
&\leq 1 + \frac{p_{\max}}{2p_{\max} \varepsilon^{-1} - p_{\max}} = 1 + \frac{p_{\max}}{p_{\max} \varepsilon^{-1} + \underbrace{(\varepsilon^{-1} - 1) p_{\max}}_{\geq 0}} \\
&\leq 1 + \varepsilon
\end{aligned}$$

The cases $f(\cdot) = \text{OPT}^\downarrow(\cdot)$ and $f(\cdot) = \text{OPT}^\uparrow(\cdot)$ follow directly via [Definition 11](#) and the inequality on OPT . The second inequality can be shown by using the first with $\nu \leftarrow \nu - \mathbf{e}^{(j)}$ and inverting both sides. $\square$

While there is nothing inherently “forcing” us to only recolor machines of critical type, doing so seems quite natural. As it simplifies the analysis and the design of our algorithms (there are simply fewer cases to consider), we capture this idea in an invariant:

Invariant 4.A (Recoloring Invariant). At a step $q \rightarrow q+1$, only machines $i \in [m]$ of critical type are recolored.

We are now ready to describe how we want to modify a state each time we add or remove a job, such that the new state has distance of solutions bounded in $\text{poly}(\varepsilon^{-1}, p_{\max})$.

5.4.2 Permanent Case: Handling Insertion

We begin by looking at the permanent setting, where jobs can only be inserted. The change of parameters $\mu^{(\text{blue})}$ and α depends on how much we have to increase the κ -freeness of the current state to “fit in” the new job. First, we make a general observation about solutions with enough κ -freeness.

Lemma 22 (Adding a job to a κ -free state). *Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ be valid and κ -free. Then for any $\nu^\Delta \in \mathbb{N}_{\geq 0}^d$ with $p^\top \nu^\Delta \leq \kappa$, $(\alpha, \mu^{(\text{blue})}, \nu + \nu^\Delta)$ is $\text{OPT}^\uparrow(\nu)$ -valid.*

Proof. Let $S' := (\alpha - \kappa, \mu^{(\text{blue})}, \nu)$. Then S' is valid by [Definition 13](#). Let $(x, \nu^{(\text{blue})})$ be the solution associated with S' . If we now increase the usable blue area back to α , this solution remains feasible and has

unchanged makespan. The knapsack constraint then has slack at least κ , so adding the job set ν^Δ can be done by simply placing the jobs onto the blue machines. The new solution does not violate [Invariant 3.B](#) nor [Invariant 3.A](#), as its makespan is no larger than the makespan of $(x, \nu^{(\text{blue})})$. $\square$

So if a state is at least p_j -free, we can directly add a job of type j to the ILP without changing any of the state parameters, and still stay valid. If this is not possible, we may have increase the freeness somehow. Luckily, when coloring a machine of critical-type($\text{OPT}^\uparrow(\nu)$) red $\rightarrow$ blue, we get at least $\varepsilon s_i T = \varepsilon \ell \geq 2p_{\max}$ bonus capacity due to the definition of ℓ and the additional capacity required on blue machines by [Invariant 3.B](#), leading to the following lemma.

Lemma 23 (Recoloring a Red Machine). *If $S := (\alpha, \mu^{(\text{blue})}, \nu)$ is valid and there is a red machine of type $t \in [\tau]$ with $s_t \text{OPT}^\uparrow \geq \ell$, then $S' := (\alpha + s_t \text{OPT}^\uparrow + p_{\max}, \mu^{(\text{blue})} + \mathbb{e}^{(t)}, \nu)$ is valid and $p_{\max}$ -free. Moreover, $C_{\max}(S') \leq C_{\max}(S)$.*

Proof. Let i be the machine to be recolored, t its type, and let σ_i be the assignment of jobs to i . Then $(\alpha + s_t \text{OPT}^\uparrow(\nu), \mu^{(\text{blue})} + \mathbb{e}^{(t)}, \nu)$ has a makespan no larger than $(\alpha, \mu^{(\text{blue})}, \nu)$, as we can do the following modifications to the solution $(x, \nu^{(\text{blue})})$ associated to $(\alpha, \mu^{(\text{blue})}, \nu)$:

1. Set $x_{t, \sigma_i} \leftarrow x_{t, \sigma_i} - 1$, $\nu \leftarrow \nu - \sigma_i$ and $\mu_t \leftarrow \mu_t - 1$; then the modified solution is feasible and its makespan did not increase.
2. Set $\alpha \leftarrow \alpha + s_t \text{OPT}^\uparrow(\nu)$, $\nu \leftarrow \nu + \sigma_i$, and $\nu^{(\text{blue})} \leftarrow \nu^{(\text{blue})} + \sigma_i$; then the job set is the same as in the beginning, the makespan did not increase and the knapsack constraint is not violated as $p^\top \nu^{(\text{blue})} \leq \alpha$, so $p^\top (\nu^{(\text{blue})} + \sigma_i) \leq \alpha + p^\top \sigma_i \leq \alpha + s_t \text{OPT}^\uparrow$.
3. $\mu_t^{(\text{blue})} \leftarrow \mu_t^{(\text{blue})} + 1$ and $\mu_t \leftarrow \mu_t + 1$, which is a change of parameters, but does not change the RHS of the ILP where only $\mu - \mu^{(\text{blue})}$ appears.
4. $\alpha \leftarrow \alpha + p_{\max}$; then the state becomes $p_{\max}$ -free if it was valid before.

To prove the validity of the state before the last step, let us consider the maximally available blue area. We know exactly how much it changes:

$$\begin{aligned} (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} + \mathbb{e}^{(t)}) - (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} \\ = (1 + \varepsilon) s_t \text{OPT}^\uparrow. \end{aligned}$$

Thus, by subtracting $(1 + \varepsilon) s_t \text{OPT}^\uparrow$ from each side, we have

$$\begin{aligned} \alpha &\leq (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} \\ &= (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} + \mathbb{e}^{(t)}) - (1 + \varepsilon) s_t \text{OPT}^\uparrow \\ &\iff \alpha + (1 + \varepsilon) s_t \text{OPT}^\uparrow \leq (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} + \mathbb{e}^{(t)}) . \end{aligned}$$

As $s_t \text{OPT}^\uparrow + p_{\max} \leq (1 + \varepsilon) s_t \text{OPT}^\uparrow$ by the definition of ℓ , α is increased by exactly $s_t \text{OPT}^\uparrow + p_{\max}$ after the last step of our procedure, which implies that afterwards, [Invariant 3.B](#) is not violated for $T \leftarrow \text{OPT}^\uparrow$. Since before the last step, the value of α was even smaller, we get that in both situations, the current state is indeed valid. Hence, $(\alpha + s_t \text{OPT}^\uparrow + p_{\max}, \mu^{(\text{blue})} + \mathbf{e}^{(t)}, \nu)$ is $p_{\max}$ -free by [Definition 13](#), and has makespan at most $C_{\max}(S)$, which eventually shows the claim. $\square$

By [Invariant 4.A](#) we may only recolor machines of critical type $\text{red} \rightarrow \text{blue}$ to gain $p_{\max}$ -freeness. What if all machines of this type have already been recolored? We then can make the following observation about the validity and makespan of such solutions.

Lemma 24. *Let ν be a job set and for all $t \in [\tau]$, $\mu_t^{(\text{blue})} = \mu_t$ if $s_t \text{OPT}^\uparrow \geq \ell$ and 0 otherwise. Then $\left(\left\lfloor (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} \right\rfloor, \mu^{(\text{blue})}, \nu \right)$ is valid and its makespan is at most OPT .*

Proof. At first glance it might seem like a contradiction that the makespan could be less than OPT . But this actually can happen, as [Definition 11](#) differs from [Definition 12](#) in terms of the parameterization of the invariants. In [Definition 11](#), they have to hold for OPT , while for a valid solution they are only required to hold for $\text{OPT}^\uparrow$. Since our definition of makespan of a state only considers $C_{\max}(x)$ of the associated solution $(x, \nu^{(\text{blue})})$, we can indeed have a makespan below OPT . Let $\bar{S} := (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \nu)$ be an OPT -valid state, i.e. such that the invariants [3.A](#) and [3.B](#) both hold for $T \leftarrow \text{OPT}$, and the associated solution has makespan at most OPT . Such a state always exists by definition of OPT . Then

$$\begin{aligned} \bar{\alpha} &\leq (1 + \varepsilon) \text{OPT}(\nu) s^\top \bar{\mu}^{(\text{blue})} \leq (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \bar{\mu}^{(\text{blue})} \\ &= (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} - (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} - \bar{\mu}^{(\text{blue})}) . \end{aligned}$$

Note that $\mu^{(\text{blue})} \geq \bar{\mu}^{(\text{blue})}$, as otherwise, by assumption on the form of $\mu^{(\text{blue})}$, under $\bar{\mu}^{(\text{blue})}$ either more than μ_t machines are colored blue for some $t \in [\tau]$, or a machine of some type $t \in [\tau]$ with $s_t \text{OPT}^\uparrow < \ell$ is colored blue, which would violate [Invariant 3.B](#). So in particular, by adding $(1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} - \bar{\mu}^{(\text{blue})})$ on each side of the inequality and rounding down, we have

$$\begin{aligned} \alpha' &:= \left\lfloor \bar{\alpha} + (1 + \varepsilon) \text{OPT}^\uparrow \cdot s^\top (\mu^{(\text{blue})} - \bar{\mu}^{(\text{blue})}) \right\rfloor \\ &\leq \left\lfloor (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} \right\rfloor . \end{aligned}$$

By inductively applying [Lemma 23](#) it follows that $C_{\max}(\alpha', \mu^{(\text{blue})}, \nu) \leq \text{OPT}$. Thus, as $\alpha' \leq \left\lfloor (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} \right\rfloor$, also

$$C_{\max}(\left\lfloor (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} \right\rfloor, \mu^{(\text{blue})}, \nu) \leq \text{OPT} .$$

$\square$

We can use this statement now to show the validity of states where a new job is added and $\text{OPT}^\uparrow$ doesn't change.

Lemma 25 (Existence of a Red Machine). *Consider $j \in [d]$. If $(\alpha, \mu^{(\text{blue})}, \nu)$ is valid and $\text{OPT}^\uparrow(\nu) = \text{OPT}^\uparrow(\nu + \mathfrak{e}^{(j)})$, then either $(\alpha + \kappa, \mu^{(\text{blue})}, \nu + \mathfrak{e}^{(j)})$ is valid for some $\kappa \in [p_j]_0$, or there is a red machine of capacity at least ℓ .*

Proof. Assume that $(\alpha + \kappa, \mu^{(\text{blue})}, \nu)$ is invalid for all $\kappa \in [p_j]_0$. Then for some $\kappa \in [p_j - 1]_0$ we have $\alpha + \kappa = \left\lfloor (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} \right\rfloor$, as otherwise, [Definition 13](#) together with [Lemma 22](#) lead to a contradiction for $\alpha \leftarrow \alpha + p_j$. If there was no red machine of capacity at least ℓ , we would have colored all machines blue that we are allowed to under [Invariant 3.A](#), and with $\alpha \leftarrow \alpha + \kappa$ we would have used every bit of area that is available under [Invariant 3.B](#). This is a contradiction to [Lemma 24](#). $\square$

To now prove correctness and bounded distance, we need the following proposition.

Proposition 4 (Increasing α for valid states). *If $S = (\alpha, \mu^{(\text{blue})}, \nu)$ is valid, then $S' = (\alpha', \mu^{(\text{blue})}, \nu)$ is valid for any $\alpha' \in [\alpha; (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})}]$.*

Proof. Let $(x, \nu^{(\text{blue})})$ be the solution associated to S . Clearly, $(x, \nu^{(\text{blue})})$ is a feasible solution to S' . Moreover, $C_{\max}(x) \leq \text{OPT}^\uparrow$ as S is valid, and since $\text{OPT}^\uparrow$ is the same for S' (as it only depends on ν), S' is valid. $\square$

We are now ready to describe our algorithm for inserting a new job for a state S , such that the new job set is valid to the returned state, and the distance from S to the returned state is bounded. This algorithm is called `INSERT`, and can be found in [Algorithm 10](#). It follows directly from the definition of the algorithm that `INSERT` does not violate [Invariant 4.A](#).

We rephrase the algorithm as a lemma, and directly show that it always computes a valid state, and that the resulting state has parameters whose difference to the parameters of the input state is bounded in $\text{poly}(p_{\max}) \leq \text{poly}(\varepsilon^{-1}, p_{\max})$. This will then significantly simplify the proofs of further properties.

Lemma 26 (`INSERT` Rephrased). *Given a valid state $S = (\alpha, \mu^{(\text{blue})}, \nu)$ and $j \in [d]$, the call `INSERT(j, S)`*

- (i) *if $(\alpha, \mu^{(\text{blue})}, \nu + \mathfrak{e}^{(j)})$ is valid, returns $(\alpha, \mu^{(\text{blue})}, \nu + \mathfrak{e}^{(j)})$ if $\text{OPT}^\uparrow(\nu) = \text{OPT}^\uparrow(\nu + \mathfrak{e}^{(j)})$, and $((1 + \varepsilon) \text{OPT}^\downarrow(\nu + \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})}), \mu^{(\text{blue})}, \nu + \mathfrak{e}^{(j)})$ else, where $(1 + \varepsilon) \text{OPT}^\downarrow(\nu + \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} \leq \alpha + p_j$; otherwise*
- (ii) *returns a valid state $(\bar{\alpha}, \mu^{(\text{blue})}, \nu + \mathfrak{e}^{(j)})$ with $\bar{\alpha} \in (\alpha; \alpha + 2p_j]$; or otherwise*

Algorithm 10 Inserting a new job of type j , given the current state S

```

1: function INSERT( $j, S$ )
2:    $S$  is of the form  $(\alpha, \mu^{(\text{blue})}, \nu)$ 
3:   if  $\text{OPT}^\uparrow(\nu + \mathbb{e}^{(j)}) = \text{OPT}^\uparrow(\nu)$  then
4:     if  $S$  is not  $p_j$ -free then
5:        $\alpha \leftarrow \min \left\{ (1 + \varepsilon) \text{OPT}^\uparrow(\nu + \mathbb{e}^{(j)}) s^\top \mu^{(\text{blue})}, \alpha + p_j \right\}$ 
6:       if  $S$  is (still) not  $p_j$ -free then
7:          $t \leftarrow \text{critical-type}(\text{OPT}^\uparrow(\nu))$   $\triangleright$  Note that  $t \neq \perp$ 
8:          $\mu^{(\text{blue})} \leftarrow \mu^{(\text{blue})} + \mathbb{e}^{(t)}$   $\triangleright$  Note that  $\mu_t^{(\text{blue})} \neq \mu_t$ 
9:          $\alpha \leftarrow \alpha + s_t \text{OPT}^\uparrow(\nu) + p_{\max}$ 
10:      end if
11:    end if
12:  else
13:    if  $(\alpha, \mu^{(\text{blue})}, \nu + \mathbb{e}^{(j)})$  is valid then
14:       $\alpha \leftarrow (1 + \varepsilon) \text{OPT}^\downarrow(\nu + \mathbb{e}^{(j)}) s^\top \mu^{(\text{blue})}$   $\triangleright$  Only needed for
REMOVE
15:    else
16:       $\alpha \leftarrow \alpha + p_j$ 
17:    end if
18:  end if
19:  return  $(\alpha, \mu^{(\text{blue})}, \nu + \mathbb{e}^{(j)})$ 
20: end function

```

(iii) correctly concludes that $(\bar{\alpha}, \mu^{(\text{blue})}, \nu + \mathbb{e}^{(j)})$ is invalid for all $\bar{\alpha}$, and returns a valid state of the form $(\bar{\alpha}, \mu^{(\text{blue})} + \mathbb{e}^{(\text{critical-type}(\text{OPT}^\uparrow(\nu)))}, \nu + \mathbb{e}^{(j)})$ where $\bar{\alpha} < \alpha + \ell + p_{\max} + p_j$.

Proof. In the following, we can assume that there always exists a machine i with $s_i = s_{\max}$, and $s_{\max} \text{OPT}^\uparrow(\nu + \mathbb{e}^{(j)}) \geq \ell$, as otherwise, it holds for both the input and return state that $\alpha = 0$, $\mu^{(\text{blue})} = \mathbf{0}$ by [Invariant 3.A](#), and thus, (i) is always fulfilled.

Case 1: $(\alpha, \mu^{(\text{blue})}, \nu + \mathbb{e}^{(j)})$ is valid.

Case 1.1: $\text{OPT}^\uparrow(\nu + \mathbb{e}^{(j)}) = \text{OPT}^\uparrow(\nu)$. Then the if-branch of the first if-statement is entered and neither α nor $\mu^{(\text{blue})}$ are changed. Since the algorithm then returns $(\alpha, \mu^{(\text{blue})}, \nu + \mathbb{e}^{(j)})$, this shows the claim, as (i) is satisfied.

Case 1.2: $\text{OPT}^\uparrow(\nu + \mathbb{e}^{(j)}) > \text{OPT}^\uparrow(\nu)$. Then the else-branch of the first if statement is entered. Within this branch, the first if-branch of the next if-statement is entered due to the assumption of Case 1. Thus, the algorithm eventually returns

$$\left((1 + \varepsilon) \text{OPT}^\downarrow(\nu + \mathbb{e}^{(j)}) s^\top \mu^{(\text{blue})}, \mu^{(\text{blue})}, \nu + \mathbb{e}^{(j)} \right).$$

By [Lemma 21](#) and using the fact that $\text{OPT}^\uparrow$ and $\text{OPT}^\downarrow$ are restricted to integer powers of $(1 + \varepsilon)$,

$$(1 + \varepsilon) \text{OPT}^\downarrow(v + \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} = (1 + \varepsilon) \text{OPT}^\uparrow(v) s^\top \mu^{(\text{blue})} \geq \alpha,$$

where the latter follows from the fact that the initial state was valid and thus satisfied [Invariant 3.B](#). Now, if $(1 + \varepsilon) \text{OPT}^\downarrow(v + \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} = (1 + \varepsilon) \text{OPT}^\uparrow(v) s^\top \mu^{(\text{blue})} > \alpha + p_j$, then the state $((1 + \varepsilon) \text{OPT}^\uparrow(v) s^\top \mu^{(\text{blue})}, \mu^{(\text{blue})}, v + \mathfrak{e}^{(j)})$ would be valid for $\text{OPT}^\uparrow(v)$ by [Lemma 22](#), and thus $\text{OPT}^\uparrow(v) = \text{OPT}^\uparrow(v + \mathfrak{e}^{(j)})$, a contradiction to the assumption of Case 1.2. Thus, $(1 + \varepsilon) \text{OPT}^\downarrow(v + \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} \leq \alpha + p_j$. Since $\mu^{(\text{blue})}$ remains unchanged, this shows the claim, as (i) is satisfied.

Case 2: $(\alpha, \mu^{(\text{blue})}, v + \mathfrak{e}^{(j)})$ is invalid and $\text{OPT}^\uparrow(v + \mathfrak{e}^{(j)}) \neq \text{OPT}^\uparrow(v)$. Then only [Line 16](#) is executed. By definition of this line, α is increased by at most p_j . The returned state is valid: By the assumption, we know that $\text{OPT}^\uparrow(v + \mathfrak{e}^{(j)}) > \text{OPT}^\uparrow(v)$ and that there exists a machine i with speed $s_{\max}$ such that $s_{\max} \text{OPT}(v + \mathfrak{e}^{(j)}) \geq \ell$. This machine i 's capacity is then increased by at least $2p_{\max}$, as here, $\text{OPT}^\uparrow(v + \mathfrak{e}^{(j)}) = (1 + \varepsilon) \text{OPT}^\uparrow(v)$ by definition of $\text{OPT}^\uparrow$, and therefore, i 's capacity is increased by at least $(1 + \varepsilon)^{-1} \ell \geq 2p_{\max}$, since i had capacity at least $(1 + \varepsilon)^{-1} \ell$ before by [Lemma 21](#).

Thus, if i is colored blue, [Invariant 3.B](#) is maintained. If i is colored red instead, j can be inserted without changing α at all, and the state $(\alpha, \mu^{(\text{blue})}, v + \mathfrak{e}^{(j)})$ would be valid, a contradiction to the assumption of Case 2. It follows that (ii) is satisfied.

Case 3: $(\alpha, \mu^{(\text{blue})}, v + \mathfrak{e}^{(j)})$ is invalid and $\text{OPT}^\uparrow(v + \mathfrak{e}^{(j)}) = \text{OPT}^\uparrow(v)$. Then the second if-branch is entered because of [Lemma 22](#).

Case 3.1: there is a valid state of the form $(\tilde{\alpha}, \mu^{(\text{blue})}, v + \mathfrak{e}^{(j)})$. Let α' be the value of α after [Line 5](#). We prove that (ii) will always be satisfied.

Case 3.1.1: $\alpha' = \alpha + p_j$. Then $(\alpha', \mu^{(\text{blue})}, v)$ is p_j -free by [Definition 13](#) and hence $(\alpha', \mu^{(\text{blue})}, v)$ is valid by [Lemma 22](#). Moreover, $\tilde{\alpha} := \alpha' = \alpha + p_j$ satisfies the condition of (ii) on $\tilde{\alpha}$.

Case 3.1.2: $\alpha' = (1 + \varepsilon) \text{OPT}^\uparrow(v + \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} < \alpha + p_j$. Then, with α' , we use all the space available to us. By our case (Case 3.1), we know that there is a valid state of the form $(\tilde{\alpha}, \mu^{(\text{blue})}, v + \mathfrak{e}^{(j)})$. Since α' is the largest possible value (otherwise [Invariant 3.B](#) is violated), $\alpha' \geq \tilde{\alpha}$. By [Proposition 4](#), $(\alpha', \mu^{(\text{blue})}, v + \mathfrak{e}^{(j)})$ is valid. Setting $\tilde{\alpha} := \alpha' < \alpha + p_j$ thus shows the claim.

Case 3.2: $(\tilde{\alpha}, \mu^{(\text{blue})}, v + \mathfrak{e}^{(j)})$ is invalid for all $\tilde{\alpha}$. Then the second if-statement is entered with usable blue area $\alpha' = (1 + \varepsilon) \text{OPT}^\uparrow(v + \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} < \alpha + p_j$ (otherwise, $(\alpha', \mu^{(\text{blue})}, v)$ was p_j -free by [Definition 13](#) $\nmid$). We show that (iii) is satisfied.

Case 3.2.1: $t \neq \perp \wedge \mu_t^{(\text{blue})} \neq \mu_t$. Then $\bar{\alpha} := \alpha' + \ell + p_{\max}$ and the state $(\bar{\alpha}, \mu^{(\text{blue})} + \mathbb{e}^{(t)}, \nu + \mathbb{e}^{(j)})$ is returned by our algorithm, which is valid by Lemma 22 as $(\bar{\alpha}, \mu^{(\text{blue})} + \mathbb{e}^{(t)}, \nu)$ is $p_{\max}$ -free.

Case 3.2.2: $t = \perp \vee \mu_t^{(\text{blue})} = \mu_t$. Then INSERT would fail, as it tries to recolor a machine of undefined type or a machine of a type where all machines are blue already. We show that this case cannot occur. First, we know that, since $\text{OPT}^\uparrow(\nu) = \text{OPT}^\uparrow(\nu + \mathbb{e}^{(j)})$, in this case there is at least one red machine by Lemma 25. By Invariant 4.A this implies that at some point before, a job of type j' was added which lead to an increase of $\text{OPT}^\uparrow$ while there was still a red machine with capacity $s_{j'} \text{OPT}^\uparrow = \ell$. This leads to a contradiction, as, if at that step, this machine was recolored red $\rightarrow$ blue, then j' could have been added without increasing $\text{OPT}^\uparrow$ by Lemma 23. $\square$

As an immediate consequence, since $(1 + \varepsilon)\ell \leq \text{poly}(\varepsilon^{-1}, p_{\max})$, we get that INSERT always computes a valid state $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ such that $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 + \|\Delta\nu\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max})$.

5.4.3 Dynamic Case: Handling Removal

After successfully having constructed a robust algorithm for the static case, we now move on to the dynamic setting, for which we also need to be able to handle the removal of jobs. We make use of the subroutine TRYBTR (Algorithm 11) that simply tries to recolor up to k machines blue $\rightarrow$ red without violating Invariant 3.B. We then propose algorithm REMOVE, to be found in Algorithm 12, as a procedure to remove a job, using TRYBTR as a subroutine.

Algorithm 11 Recoloring subroutine

```

1: function TRYBTR( $k, \alpha, \mu^{(\text{blue})}, \nu$ )
2:   if  $\mu^{(\text{blue})} \neq \mathbf{0}$  then
3:      $t \leftarrow \text{critical-type}(\text{OPT}^\uparrow(\nu))$ 
4:     if  $t \neq \perp$  then
5:        $k \leftarrow \min\{k, \mu_t^{(\text{blue})}\}$ 
6:        $\mu^{(\text{blue})} \leftarrow \mu^{(\text{blue})} - k \cdot \mathbb{e}^{(t)}$ 
7:        $\alpha \leftarrow \alpha - (1 + \varepsilon) \cdot k s_t \cdot \text{OPT}^\uparrow(\nu)$ 
8:     end if
9:   end if
10:  return  $(\alpha, \mu^{(\text{blue})}, \nu)$ 
11: end function

```

We now want to show similar properties as in the insertion case, e. g. prove that the state computed by the algorithm is valid for the new job vector, and the change of the parameters is bounded by $\text{poly}(\varepsilon^{-1}, p_{\max})$. The next lemma is almost the inverse of Lemma 23,

Algorithm 12 Removing a job of type j , given the current state S

```

1: function REMOVE( $j, S$ )
2:    $S$  is of the form  $(\alpha, \mu^{(\text{blue})}, \nu)$ 
3:    $t \leftarrow \text{critical-type}(\text{OPT}^\uparrow(\nu - \mathbb{e}^{(j)}))$ 
4:   if  $\text{OPT}^\uparrow(\nu - \mathbb{e}^{(j)}) < \text{OPT}^\uparrow(\nu)$  then
5:      $S \leftarrow \text{TRYBTR}(m, S)$  ▷ Note that this modifies  $\mu^{(\text{blue})}$ 
6:      $\alpha \leftarrow (1 + \varepsilon) \cdot \text{OPT}^\uparrow(\nu - \mathbb{e}^{(j)}) s_t \mu^{(\text{blue})}$ 
7:   end if
8:    $\nu \leftarrow \nu - \mathbb{e}^{(j)}$ 
9:   if  $S$  is  $(\varepsilon\ell + p_{\max})$ -free  $\wedge t \neq \perp \wedge \mu_t^{(\text{blue})} > 0$  then
10:     $S \leftarrow \text{TRYBTR}(1, S)$ 
11:  end if
12:  if  $S$  is  $p_{\max}$ -free  $\wedge (t = \perp \vee \mu_t^{(\text{blue})} = 0)$  then
13:     $\alpha \leftarrow \max \left\{ \alpha - p_{\max}, (1 + \varepsilon) \cdot \text{OPT}^\downarrow(\nu) \cdot s_t \mu^{(\text{blue})} \right\}$ 
14:  end if
15:  return  $(\alpha, \mu^{(\text{blue})}, \nu)$ 
16: end function

```

except that we require S to be $(\varepsilon\ell + p_{\max})$ -free instead of gaining $p_{\max}$ -freeness.

Lemma 27 (Recoloring a Blue Machine). *If $S := (\alpha, \mu^{(\text{blue})}, \nu)$ is $(\varepsilon\ell + p_{\max})$ -free and $\mu_i^{(\text{blue})} > 0$, where $i = \text{critical-type}(\text{OPT}^\uparrow(\nu))$, then $S' := (\alpha - (1 + \varepsilon)\ell, \mu^{(\text{blue})} - \mathbb{e}^{(i)}, \nu)$ is valid.*

Proof. Let i be the machine of type $i := \text{critical-type}(\text{OPT}^\uparrow(\nu))$ to be recolored. Note that $\ell = s_i \text{OPT}^\uparrow(\nu)$ by definition of $\text{critical-type}(\cdot)$. Now, let $(x, \nu^{(\text{blue})})$ be the solution associated to $\bar{S} := (\alpha - (\varepsilon\ell + p_{\max}), \mu^{(\text{blue})}, \nu)$. By [Definition 13](#) we know that $\bar{S}$ is valid. Moreover, as the job set is the same as in S , the values for $\text{OPT}^\uparrow$ also coincide. Hence, the solution $(x, \nu^{(\text{blue})})$ associated to $\bar{S}$ is a feasible solution to S where the knapsack constraint ([Constraint 5.1](#)) has slack at least $(\varepsilon\ell + p_{\max})$.

Consider an assignment of blue jobs onto blue machines such that each blue machine of type $t \in [\tau]$ is assigned jobs of total load at most $(1 + \varepsilon)s_t \text{OPT}^\uparrow(\nu) + p_{\max}$. Such an assignment can be found by a simple greedy algorithm (e.g. list-scheduling), as S is valid and thus [Invariant 3.B](#) holds for $T \leftarrow \text{OPT}^\uparrow(\nu)$. Let ν^Δ be the jobs assigned to machine i of type i . Consider the following modifications to $(x, \nu^{(\text{blue})})$ and S :

1. Set $\nu \leftarrow \nu - \nu^\Delta$, $\nu^{(\text{blue})} \leftarrow \nu^{(\text{blue})} - \nu^\Delta$; then $(x, \nu^{(\text{blue})})$ has slack $(\varepsilon\ell + p_{\max}) + p^\tau \nu^\Delta \leq (1 + \varepsilon)\ell + p_{\max}$ under the current state.
2. Set $\mu_i^{(\text{blue})} \leftarrow \mu_i^{(\text{blue})} - 1$, $\mu_i \leftarrow \mu_i - 1$, $\alpha \leftarrow \alpha - (1 + \varepsilon)\ell$ then the modified state is T -valid, as the removal of a machine together

with the jobs assigned to it cannot increase the makespan of the solution. Moreover, our solution now has slack $p_{\max} + p^\top v^\Delta - \ell$.

3. Let $v^+ \in [0..v^\Delta]$ be an inclusion minimal selection of jobs such that $p^\top(v^\Delta - v^+) \leq \ell$. Then set $x_{i,(v^\Delta - v^+)} \leftarrow x_{i,(v^\Delta - v^+)} + 1$, $\mu_i \leftarrow \mu_i + 1$, $v \leftarrow v + v^\Delta - v^+$. This solution then has makespan at most

$$\max\{T, s_i^{-1} p^\top(v^\Delta - v^+)\} \leq \max\{T, s_i^{-1} \ell\} = T.$$

Moreover, it (still) has slack $p_{\max} + p^\top v^\Delta - \ell$.

4. Note that

$$p_{\max} + p^\top v^\Delta - \ell = p_{\max} + p^\top(v^\Delta - v^+) + p^\top v^+ - \ell \geq p^\top v^+,$$

since $p^\top(v^\Delta - v^+) \geq \ell - p_{\max}$ by inclusion minimality of v^+ . Thus, we can place v^+ as blue jobs, i.e. modify $v^{(\text{blue})} \leftarrow v^{(\text{blue})} + v^+$ and $v \leftarrow v + v^+$, without violating the knapsack constraint (Constraint 5.1).

In the end, the value of v is restored, i.e. is equal to the initial value, α is decreased by $(1 + \varepsilon)\ell$, and one more machine of critical type is now colored red. Invariant 3.A cannot be violated as we only recolored a machine of the critical type i . Invariant 3.B is not violated as recolouring a machine blue $\rightarrow$ red decreases the total blue area by $(1 + \varepsilon)\ell$, but we also decreased the usable blue area α accordingly. Thus, the claim holds. $\square$

With this result in hand, we can prove the first part: Our algorithm does indeed always compute a valid state.

Lemma 28 (Correctness of REMOVE). *Given a valid state, REMOVE always computes a valid state.*

Proof. First, we show that the state computed after Line 8 is valid. Afterwards, we argue that the remaining lines will not change a valid state into an invalid one. Let j be the type of the job that is deleted, i.e. $\Delta v = e^{(j)}$. We make a case distinction on ΔOPT . If $\Delta \text{OPT}^\uparrow = 0$, then simply removing the job already yields a valid state, which is exactly what the algorithm computes. If $\Delta \text{OPT}^\uparrow < 0$, Line 5 ensures that Invariant 3.A is not violated. The line thereafter ensures that Invariant 3.B is not violated: $(1 + \varepsilon) \text{OPT}^\uparrow (v - e^{(j)})^\top \mu^{(\text{blue})}$ is exactly the blue area we get when all machines of critical-type $(\text{OPT}^\uparrow (v - e^{(j)}))$ are colored blue, which is true for the new critical-type $(\text{OPT}^\uparrow (v - e^{(j)}))$, since machines of this type had capacity $> \ell$ for $\text{OPT}^\uparrow$ and where thus colored blue by validity of the input state. For $\mu^{(\text{blue})} = \mathbb{O}$, the equality above also holds as both sides are 0. In consequence, setting α to the value above satisfies Invariant 3.B for $T \leftarrow \text{OPT}^\uparrow (v - e^{(j)})$. Validity of

the state then eventually follows from [Lemma 24](#). Since TryBTR only recolors machines of critical-type, [Invariant 4.A](#) holds as well.

Now, let us consider what happens after [Line 8](#). If a machine is recolored due to an execution of [Line 10](#), then S was $(\varepsilon\ell + p_{\max})$ -free before. The resulting state is valid also, due to [Lemma 27](#). If [Line 13](#) is executed, then α is changed by at most $p_{\max}$, but the state was $p_{\max}$ -free before, implying it is valid thereafter. $\square$

Now, in order to prove bounded migration of our proposed procedure, we first need to introduce two lower bounds on the value of α , and show that they hold for every step of our algorithm. If both lower bounds hold for our starting state, this property is implied if they hold for any execution of INSERT and REMOVE on a valid state as well. These lower bounds will help us to first prove bounded κ -freeness of most states computed by our algorithm, and by that, help us show that the “critical lines” [Line 5](#) and [Line 6](#) of REMOVE do not reduce $\mu^{(\text{blue})}$ and α too much, respectively. This then will imply bounded migration. The first lower bound is [Invariant 4.B](#).

Invariant 4.B. $i \neq \perp \implies (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} - \mathbb{e}^{(i)}) < \alpha$.

Note that $\mu^{(\text{blue})} - \mathbb{e}^{(i)}$ can have a negative entry; in this case, for the sake of simplicity, we define $(1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} - \mathbb{e}^{(j)})$ to be $-\infty$. For the initial solution with no jobs, [Invariant 4.B](#) holds as $\text{OPT} = \text{OPT}^\uparrow = 0$, implying that there cannot be a blue machine, and hence $i = \perp$. As we only use procedures INSERT and REMOVE to compute new states, if both procedure maintain the invariant, the whole algorithm does so.

Proposition 5. INSERT maintains [Invariant 4.B](#).

Proof. Since the value for α is only increased by INSERT, there are only two scenarios where the claim does not become trivial: A change of $\text{OPT}^\uparrow$ and a recolouring of a red machine (both cannot occur at the same time due to [Lemma 23](#)). Let $S := (\alpha, \mu^{(\text{blue})}, \nu)$ be the state before insertion of the new job of type j and $\tilde{S} := (\tilde{\alpha}, \tilde{\mu}^{(\text{blue})}, \tilde{\nu})$ with $\tilde{\nu} := \nu + \mathbb{e}^{(j)}$ the state afterwards.

Case 1: $\tilde{\mu}^{(\text{blue})} \neq \mu^{(\text{blue})} \wedge \text{OPT}^\uparrow(\nu) = \text{OPT}^\uparrow(\tilde{\nu})$. This means that (iii) of [Lemma 26](#) “is executed”. Then $(\alpha', \mu^{(\text{blue})}, \tilde{\nu})$ is invalid for all α' , so in particular for all $\alpha' \leq (1 + \varepsilon) \text{OPT}^\uparrow(\tilde{\nu}) s^\top \mu^{(\text{blue})}$ which are all possible values that do not violate [Invariant 3.B](#). That means that the invalidity stems from a too large makespan of the associated solutions. Recoloring a machine red $\rightarrow$ blue without increasing the α -value will only make the situation worse, i.e. cannot lead to a smaller makespan. In consequence, the state $(\tilde{\alpha}, \mu^{(\text{blue})} + \mathbb{e}^{(i)}, \tilde{\nu})$ returned from INSERT must satisfy $\tilde{\alpha} > (1 + \varepsilon) \text{OPT}^\uparrow(\tilde{\nu}) s^\top \mu^{(\text{blue})} = (1 + \varepsilon) \text{OPT}^\uparrow(\tilde{\nu}) s^\top (\tilde{\mu}^{(\text{blue})} - \mathbb{e}^{(i)})$.

Case 2: $\bar{\mu}^{(\text{blue})} = \mu^{(\text{blue})} \wedge \text{OPT}^\uparrow(\nu) \neq \text{OPT}^\uparrow(\bar{\nu})$. When using INSERT, $\text{OPT}^\uparrow(\nu) \neq \text{OPT}^\uparrow(\bar{\nu})$ implies $\text{OPT}^\uparrow(\nu) < \text{OPT}^\uparrow(\bar{\nu})$. Let $\bar{t} := \text{OPT}^\uparrow(\bar{\nu})$ be the (new) critical type. In particular, we have $\bar{t} \neq t$. Due to Definition 14 and Invariant 3.A, we either have $\bar{t} = \perp$ or $\mu_{\bar{t}}^{(\text{blue})} = 0$ directly after an increase of $\text{OPT}^\uparrow$. In the first case, the premise of the implications of Invariant 4.B evaluates to “false”, and Invariant 4.B holds. In the latter case, $(1 + \varepsilon) \text{OPT}^\uparrow(\bar{\nu}) s^\top (\mu^{(\text{blue})} - \mathbb{e}^{(\bar{t})})$ evaluates to $-\infty$, and Invariant 4.B holds as well, for all possible α . $\square$

Proposition 6. TRYBTR maintains Invariant 4.B.

Proof. Consider a call TRYBTR($k, \alpha, \mu^{(\text{blue})}, \nu$). We may wlog. assume that $t = \text{critical-type}(\text{OPT}^\uparrow) \neq \perp$, otherwise the function simply returns the input state. Moreover, we may wlog. assume that $k \leq \mu_t^{(\text{blue})}$ as internally k is modified accordingly (for any $k' > \mu_t^{(\text{blue})}$ the function behaves identically). Now, assume that Invariant 4.B holds for the initial state, i.e.

$$\alpha > (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} - \mathbb{e}^{(t)}) ,$$

and let $(\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \nu)$ be the state returned by the function. Then

$$\bar{\alpha} = \alpha - k(1 + \varepsilon) s_t \text{OPT}^\uparrow = \alpha - k(1 + \varepsilon) \ell .$$

Moreover,

$$\begin{aligned} (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \bar{\mu}^{(\text{blue})} &= (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} - k \mathbb{e}^{(t)}) \\ &= (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} - k(1 + \varepsilon) s_t \text{OPT}^\uparrow \\ &= (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} - k(1 + \varepsilon) \ell . \end{aligned}$$

We can then show the claim, i.e. given that Invariant 4.B did hold initially prove that it holds afterwards.

$$\begin{aligned} \alpha &> (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} - \mathbb{e}^{(t)}) \\ \iff \alpha - k(1 + \varepsilon) \ell &> (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} - \mathbb{e}^{(t)}) - k(1 + \varepsilon) \ell \\ \iff \bar{\alpha} &> (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} - (k + 1) \mathbb{e}^{(t)}) \\ &= (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\bar{\mu}^{(\text{blue})} - \mathbb{e}^{(t)}) \end{aligned}$$

$\square$

Proposition 7. REMOVE maintains Invariant 4.B

Proof. Since the value for α is only decreased by REMOVE, there are only two scenarios where the claim does not become trivial: A change of $\text{OPT}^\uparrow$ or a recolouring of a blue machine. Let $S := (\alpha, \mu^{(\text{blue})}, \nu)$ be the state before removal of job of type j and $\bar{S} := (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ with $\bar{\nu} := \nu - \mathbb{e}^{(j)}$ the state afterwards.

Case 1: $\bar{\mu}^{(\text{blue})} \neq \mu^{(\text{blue})} \wedge \text{OPT}^\uparrow(v) = \text{OPT}^\uparrow(\bar{v})$. The invariant can only be violated due to a decrease of α . As in this case, [Line 5](#) and [Line 6](#) are not executed, this can only be due to [Line 10](#) or [Line 13](#). The execution of [Line 10](#) maintains the invariant by [Proposition 6](#). Let $i := \text{critical-type}(\text{OPT}^\uparrow(v))$. [Line 13](#) is only executed if $i = \perp$ or $\mu_i = 0$; in both cases, the invariant trivially holds, as either its premise is false or the LHS of the inequality evaluates to $-\infty$.

Case 2: $(\bar{\mu}^{(\text{blue})} = \mu^{(\text{blue})} \vee \bar{\mu}^{(\text{blue})} \neq \mu^{(\text{blue})}) \wedge \text{OPT}^\uparrow(v) \neq \text{OPT}^\uparrow(\bar{v})$.

Then, [Line 5](#) and [Line 6](#) are executed. Again, the execution of [Line 5](#) maintains the invariant by [Proposition 6](#).

Let $\check{i} := \text{critical-type}(\text{OPT}^\uparrow(\bar{v}))$. Then, after [Line 6](#), if $\mu_i^{(\text{blue})} > 0$,

$$\tilde{\alpha} := (1 + \varepsilon) s^\top \text{OPT}^\uparrow(\bar{v}) \bar{\mu}^{(\text{blue})} > (1 + \varepsilon) \text{OPT}^\uparrow(\bar{v}) s^\top (\bar{\mu}^{(\text{blue})} - e^{(\check{i})}) .$$

Otherwise, if $\mu_i^{(\text{blue})} = 0$ or $\check{i} = \perp$, the invariant trivially holds as argued above. Thus, [Invariant 4.B](#) always holds when [Line 6](#) is executed.

In the current case of $\text{OPT}^\uparrow(v) \neq \text{OPT}^\uparrow(\bar{v})$, $\bar{S}$ can not be $p_{\max}$ -free (nor $(\varepsilon\ell + p_{\max})$ -free), as otherwise, we could simply reinsert j without a makespan increase, implying $\text{OPT}^\uparrow(v) = \text{OPT}^\uparrow(\bar{v})$, a contradiction. Thus, neither [Line 10](#) nor [Line 13](#) are executed, no more changes to $\tilde{\alpha}$ are made, and [Invariant 4.B](#) always holds. $\square$

Our second lower bound is [Invariant 4.C](#).

Invariant 4.C. $\delta := (1 + \varepsilon) \text{OPT}^\downarrow(v) s^\top (\mu^{(\text{blue})} - \mu_i^{(\text{blue})} e^{(i)}) \leq \alpha$.

As $\alpha = \delta = 0$ initially, this lower bound already holds for our starting state. Luckily, showing that INSERT or REMOVE maintain [Invariant 4.C](#) is less involved.

Proposition 8. INSERT maintains [Invariant 4.C](#).

Proof. If $\text{OPT}^\uparrow(v + e^{(j)}) = \text{OPT}^\uparrow(v)$, INSERT only increases α , while δ does not change. Else if $\text{OPT}^\uparrow(v + e^{(j)}) \neq \text{OPT}^\uparrow(v)$, α is set to $(1 + \varepsilon) \text{OPT}^\downarrow(v + e^{(j)}) s^\top \mu^{(\text{blue})} \geq \delta$ if the input state is valid already, or α is increased by at least p_j if it is not. In the latter case, $\alpha > (1 + \varepsilon) \text{OPT}^\downarrow(v + e^{(j)}) s^\top \mu^{(\text{blue})} \geq \delta$ after the increase, as otherwise, the resulting state would be valid for $\alpha = (1 + \varepsilon) \text{OPT}^\downarrow(v + e^{(j)}) s^\top \mu^{(\text{blue})} \geq \delta$ as well, a contradiction to the assumption that it was not. $\square$

Proposition 9. REMOVE maintains [Invariant 4.C](#).

Proof. If $\text{OPT}^\uparrow(v - e^{(j)}) < \text{OPT}^\uparrow(v)$, α is set to at least δ in [Line 6](#), as all machines of $\text{critical-type}(\text{OPT}^\uparrow(v))$ are colored blue $\rightarrow$ red by [Line 5](#), and all machines of the new $\text{critical-type}(\text{OPT}^\uparrow(v - e^{(j)}))$ are still colored blue in the resulting state by virtue that [Invariant 3.A](#) was maintained for the input state. While recolouring a blue machine in

Line 10, only the capacity of the recolored machine is deducted from α , maintaining $\alpha \geq \delta$ by maintaining **Invariant 4.B**. If **Line 13** is executed, α is set to at least δ as well. $\square$

Using these bounds given to us by these two invariants, we can prove upper bounds for the κ -freeness of states after the computation of a new state. We first consider **INSERT**.

Proposition 10. *Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ be a valid state and κ maximal such that S is κ -free, and $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \nu + \mathfrak{e}^{(j)})$ be a $\bar{\kappa}$ -free state returned by **INSERT**(j, S). Let $\bar{i} := \text{critical-type}(\text{OPT}^\uparrow(\nu + \mathfrak{e}^{(j)}))$ and $\bar{\delta} := (1 + \varepsilon) \text{OPT}^\downarrow(\nu + \mathfrak{e}^{(j)})_{S^\top}(\mu^{(\text{blue})} - \mu_i^{(\text{blue})} \mathfrak{e}^{(i)})$. Then*

$$\bar{\alpha} > \alpha \wedge \bar{\alpha} > \bar{\delta} \implies \bar{\kappa} \leq \ell + 2p_{\max} - 1 .$$

Proof. Consider **Lemma 26**; read the 3 cases as statements of an algorithm. If the algorithm terminates after **(i)**, either $\bar{\alpha} = \alpha$ or $\bar{\alpha} = (1 + \varepsilon) \text{OPT}^\downarrow(\nu + \mathfrak{e}^{(j)})_{S^\top} \mu^{(\text{blue})} = \bar{\delta}$, where the last equality follows from the fact that no machines of type $\bar{i}$ are colored blue in this case, implying $\mu_{\bar{i}}^{(\text{blue})} = 0$, and the statement is always true as its LHS evaluates to false. If the algorithm terminates after **(ii)**, then $\bar{\kappa} < p_j$, as we added up to $2p_j$ usable blue area, but immediately occupy p_j of it by the job of type j . Now assume that it terminates after **(iii)**. Then, by the inequality $\bar{\alpha} < (1 + \varepsilon)\ell + p_j$, the statement follows directly, as this case only occurs if S was less than p_j -free before, p_j space is immediately occupied by the new job of type j , and thus it must be less than $(\ell + p_{\max} + p_j)$ -free after. $\square$

We show a similar statement for the slack of α after using **REMOVE**.

Proposition 11. *Let S be a valid state and κ maximal such that S is κ -free, $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ be a $\bar{\kappa}$ -free state returned by **REMOVE**(j, S). Let $\bar{i} := \text{critical-type}(\text{OPT}^\uparrow(\bar{\nu}))$ and $\bar{\delta} := (1 + \varepsilon) \text{OPT}^\downarrow(\bar{\nu})_{\bar{S}^\top}(\mu^{(\text{blue})} - \mu_{\bar{i}}^{(\text{blue})} \mathfrak{e}^{(\bar{i})})$. Then*

$$(i) \quad \bar{\alpha} > \bar{\delta} \implies \bar{\kappa} \leq \max \{ \kappa, (\varepsilon\ell + p_{\max}) \} .$$

$$(ii) \quad \text{OPT}^\uparrow(\bar{\nu} - \mathfrak{e}^{(j)}) < \text{OPT}^\uparrow(\bar{\nu}) \implies \bar{\kappa} < p_{\max} .$$

Proof. **(i)** If $\bar{\alpha} > \bar{\delta}$, then $\text{OPT}^\uparrow(\bar{\nu}) = \text{OPT}^\uparrow(\bar{\nu} - \mathfrak{e}^{(j)})$, as otherwise, **Line 5** and **Line 6** would have been executed and set $\alpha = (1 + \varepsilon) \text{OPT}^\uparrow(\bar{\nu} - \mathfrak{e}^{(j)})_{\bar{S}^\top}(\mu^{(\text{blue})} - \mu_{\bar{i}}^{(\text{blue})} \mathfrak{e}^{(\bar{i})}) = \bar{\delta}$, where the last equality follows from **Proposition 3**. If $\bar{\kappa} < (\varepsilon\ell + p_{\max})$, the claim directly holds. Otherwise, $\bar{\kappa} \geq (\varepsilon\ell + p_{\max})$. We backtrack which lines have been executed within **REMOVE**: If $\bar{S}$ is $(\varepsilon\ell + p_{\max})$ -free even after **REMOVE**, either **Line 10** or **Line 13** have been executed during this call, as both lines can only reduce the freeness, and thus, the state before these lines was at least $(\varepsilon\ell + p_{\max})$ -free as well, and one of the conditions before the respective line is then always true. Since $\bar{\alpha} > \bar{\delta}$ by assumption, and

$\delta = (1 + \varepsilon) \text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})}$ if $i = \perp$ or $\mu_i^{(\text{blue})} = 0$, each of both lines reduce α , and thus freeness, by at least $p_{\max}$ if they are executed, so the state S' before these lines is at least $\bar{\kappa} + p_{\max}$ -free. Since S' is derived from S by simply removing some job, we can conclude that S is at least $\bar{\kappa}$ -free, which shows the claim.

(ii) $\text{OPT}^\uparrow(\nu - \mathfrak{e}^{(j)}) < \text{OPT}^\uparrow(\nu)$ holds by assumption. If $\bar{S}$ was at least $p_{\max}$ -free, reinserting a job of type j would yield a state S' with $\text{OPT}^\uparrow(\nu - \mathfrak{e}^{(j)}) = \text{OPT}^\uparrow(\nu)$ by [Lemma 22](#), a contradiction. $\square$

Consequently, we combine the two previous statements to upper bound the slack of all states computed by our algorithm using only INSERT and REMOVE to compute new states.

Proposition 12. *Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ be some valid state, and let $i := \text{critical-type}(\text{OPT}^\uparrow(\nu))$ and $\delta := (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\mu^{(\text{blue})} - \mu_i^{(\text{blue})} \mathfrak{e}^{(i)})$.*

Let $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ be a state computed either by $\bar{S} = \text{INSERT}(j, S)$ or $\bar{S} = \text{REMOVE}(j, S)$. Let $\tilde{i} := \text{critical-type}(\text{OPT}^\uparrow(\bar{\nu}))$ and $\bar{\delta} := (1 + \varepsilon) \text{OPT}^\downarrow(\bar{\nu}) s^\top (\bar{\mu}^{(\text{blue})} - \bar{\mu}_{\tilde{i}}^{(\text{blue})} \mathfrak{e}^{(\tilde{i})})$. Then

$$\begin{aligned} (\alpha > \delta &\implies S \text{ is at most } (\ell + 2p_{\max} - 1)\text{-free}) \\ \implies (\bar{\alpha} > \bar{\delta} &\implies \bar{S} \text{ is at most } (\ell + 2p_{\max} - 1)\text{-free}) . \end{aligned}$$

Proof. In the following, we assume that the LHS of the implication $(\alpha > \delta \implies S \text{ is at most } (\ell + 2p_{\max} - 1)\text{-free})$ as well as the LHS of the implication on the RHS $(\bar{\alpha} > \bar{\delta})$ both evaluate to true, as otherwise, the implication always evaluates to true. We make a case distinction regarding $\text{OPT}^\uparrow(\bar{\nu})$ in relation to $\text{OPT}^\uparrow(\nu)$.

Case 1: $\text{OPT}^\uparrow(\bar{\nu}) > \text{OPT}^\uparrow(\nu)$. This only occurs if $\bar{S} = \text{INSERT}(j, S)$. Then $\bar{\alpha} > \alpha$, as, if $\bar{\alpha}$ is not increased, $\bar{\alpha} = \bar{\delta}$ by [Line 14](#) of INSERT and the fact that all machines of $\tilde{i}$ are colored red in this case, a contradiction to the assumption that $\bar{\alpha} > \bar{\delta}$. Thus, $\bar{S}$ is at most $(\ell + 2p_{\max} - 1)$ -free by [Proposition 10](#).

Case 2: $\text{OPT}^\uparrow(\bar{\nu}) < \text{OPT}^\uparrow(\nu)$. This only occurs if $\bar{S} = \text{REMOVE}(j, S)$. Then $\bar{S}$ is at most $(\ell + 2p_{\max} - 1)$ -free by [Proposition 11](#), (ii).

Case 3: $\text{OPT}^\uparrow(\bar{\nu}) = \text{OPT}^\uparrow(\nu)$.

Case 3.1: $\bar{S} = \text{INSERT}(j, S)$ It holds that $\bar{\alpha} \geq \alpha$ by definition of INSERT. If $\bar{\alpha} = \alpha$, then the statement is true: Either $\bar{\alpha} = \alpha = \delta$, which makes the statement always true, or $\bar{\alpha} = \alpha > \delta$, implying $(\ell + 2p_{\max} - 1)$ -freeness for S , and thus $\bar{S}$ is at most $(\ell + 2p_{\max} - 1)$ -free since insertion a job without increasing α or $\text{OPT}^\uparrow$ can only decrease the freeness.

Else if $\bar{\alpha} > \alpha$, the statement is also true, as $\alpha \geq \delta$ by [Invariant 4.C](#) and $\delta = \bar{\delta}$ when $\text{OPT}^\uparrow(\bar{\nu}) = \text{OPT}^\uparrow(\nu)$, and thus $\bar{S}$ is at most $(\ell + 2p_{\max} - 1)$ -free by [Proposition 10](#).

Case 3.2: $\bar{S} = \text{REMOVE}(j, S)$ It holds that $\bar{\alpha} \leq \alpha$ by definition of REMOVE . Also, note that $\delta = \bar{\delta}$ when $\text{OPT}^\uparrow(\bar{\nu}) = \text{OPT}^\uparrow(\nu)$. Thus, $\bar{\alpha} \geq \bar{\delta} = \delta$, where the first inequality follows from [Invariant 4.C](#). If $\bar{\alpha} = \bar{\delta}$, the statement is always true. Else if $\alpha \geq \bar{\alpha} > \bar{\delta} = \delta$, S is at most $(\ell + 2p_{\max} - 1)$ -free by our assumptions, and [Proposition 11, \(i\)](#) then shows that $\bar{S}$ is at most $(\ell + 2p_{\max} - 1)$ -free as well, proving the statement.

This proves the claim. $\square$

Since $(\alpha > \delta \implies S \text{ is at most } (\ell + 2p_{\max} - 1)\text{-free})$ is true for our starting state ($\nu = 0$ and the state is 0-free), and we use only INSERT and REMOVE to compute new states, [Proposition 12](#) holds for every step our algorithm. This implies the following corollary.

Corollary 2. *For each state $S = (\alpha, \mu^{(\text{blue})}, \nu)$ computed by our algorithm*

$$\alpha > \delta \implies S \text{ is at most } (\ell + 2p_{\max} - 1)\text{-free}$$

is true, with $i := \text{critical-type}(\text{OPT}^\uparrow(\nu))$ and $\delta = (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\mu^{(\text{blue})} - \mu_i^{(\text{blue})} \mathbb{E}^{(i)})$.

We can use this bound on the κ -freeness of states to show the following claim on the coloring and α change caused by [Line 5](#) and [Line 6](#), respectively.

Proposition 13. *For any execution of $\text{REMOVE}(j, S)$ on a state S produced by our algorithm and a job of type j with $\nu_j > 0$ the following holds:*

- (i) *On any execution of [Line 5](#), at most $\frac{(\ell + 2p_{\max})}{p_{\max}}$ machines are recolored blue $\rightarrow$ red.*
- (ii) *On any execution of [Line 6](#), α is reduced by at most $\ell + 3p_{\max}$.*

Proof. Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ and $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \nu) = \text{REMOVE}(j, S)$. Also, let $i = \text{critical-type}(\text{OPT}^\uparrow(\nu))$ and $\delta := (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\mu^{(\text{blue})} - \mu_i^{(\text{blue})} \mathbb{E}^{(i)})$. [Line 5](#) and [Line 6](#) are only executed during $\text{REMOVE}(j, S)$ if $\text{OPT}^\uparrow(\nu - \mathbb{E}^{(j)}) < \text{OPT}^\uparrow(\nu)$. Note that, in this case, $\delta = (1 + \varepsilon) \text{OPT}^\uparrow(\nu - \mathbb{E}^{(j)}) s^\top (\mu^{(\text{blue})} - \mu_i^{(\text{blue})} \mathbb{E}^{(i)})$ by [Lemma 21](#). Thus, $\bar{\alpha} = \delta$ after execution of [Line 6](#).

$\alpha > \delta$ Then S is at most $(\ell + 2p_{\max} - 1)$ -free by [Corollary 2](#). Reinserting a job of type j with $\text{INSERT}(\bar{S}, j)$ after its removal yields a valid state $S' = (\alpha', \mu^{(\text{blue})'}, \nu)$ with $\text{OPT}^\uparrow(\nu) > \text{OPT}^\uparrow(\nu - \mathbb{E}^{(j)})$. For S' , $\mu^{(\text{blue})'}_i \leq 1$ and $\alpha' \leq \delta + p_j$, as, for $\text{OPT}^\uparrow(\nu) > \text{OPT}^\uparrow(\nu - \mathbb{E}^{(j)})$, $\text{INSERT}(\bar{S}, j)$ sets α first to δ , and then increases α by at most p_j after.

(i). Assume that [Line 5](#) is executed on state S and job of type j and more than $\frac{(\ell + 2p_{\max})}{p_{\max}}$ machines are recolored blue $\rightarrow$ red. For this to occur, $\mu_i^{(\text{blue})} \geq \frac{(\ell + 2p_{\max} + 1)}{p_{\max}}$ must hold at this point.

Then, since S' is at least 0-free due to its validity, S is at least $(\ell + 2p_{\max})$ -free, as each of the $\frac{(\ell + 2p_{\max} + 1)}{p_{\max}}$ machines that are colored blue in S but not in S' add at least $p_{\max}$ to α , due to the maintaining of [Invariant 4.B](#). This is a contradiction to S is at most $(\ell + 2p_{\max} - 1)$ -free if $\alpha > \delta$ by [Corollary 2](#).

(ii). If $\alpha > \bar{\alpha} + \ell + 3p_{\max} = \delta + \ell + 3p_{\max}$, then S is at least $(\ell + 2p_{\max})$ -free for $\mu_i^{(\text{blue})} = 0$, as $\alpha' \leq \delta + p_{\max}$ for the state S' with the same job set v . This then holds for all $\mu_i^{(\text{blue})} \geq 0$, as any additional blue machine increases the κ -freeness by at least $p_{\max}$ while maintaining [Invariant 4.B](#). This again is a contradiction to S is at most $(\ell + 2p_{\max} - 1)$ -free if $\alpha > \delta$, and thus, $\alpha \leq \delta + \ell + 3p_{\max}$, and the statement follows as $\bar{\alpha} = \delta$.

$\alpha = \delta$ (i) and (ii) are both true in this case, as $\alpha = \bar{\alpha}$, and recolouring one machine blue $\rightarrow$ red with TRYBTR reduces α by at least $(1 + \varepsilon)\ell$, implying that no machine can be recolored the case of $\alpha = \bar{\alpha}$.

$\alpha < \delta$ can never occur by [Invariant 4.C](#). $\square$

We can now finally use this result to show that REMOVE has indeed bounded migration.

Lemma 29 (REMOVE Has Bounded Migration). *Let $S = (\alpha, \mu^{(\text{blue})}, v)$ be a valid state and $j \in \text{supp}(v)$. Then for $\bar{S} := (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{v}) = \text{REMOVE}(j, S)$, $1 \leq |\Delta\alpha| + \|\Delta v\|_1 + \|\Delta\mu^{(\text{blue})}\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max})$.*

Proof. $\|\Delta v\|_1 = 1$ by definition of the problem. Due to [Proposition 13](#) (i), and the fact that in [Line 10](#), the only other blue $\rightarrow$ red recolouring, only 1 machine is recolored,

$$\|\Delta\mu^{(\text{blue})}\|_1 \leq \frac{(\ell + 2p_{\max})}{p_{\max}} + 1 \leq \text{poly}(\varepsilon^{-1}, p_{\max}).$$

Thus, α is reduced at most by

$$\text{poly}(\varepsilon^{-1}, p_{\max}) \cdot (\ell + p_{\max}) \leq \text{poly}(\varepsilon^{-1}, p_{\max})$$

by recolouring.

The only other times when α might be changed are [Line 13](#), where α is reduced by at most $p_{\max} \leq \text{poly}(\varepsilon^{-1}, p_{\max})$, and [Line 6](#). By [Proposition 13](#) (ii), the execution of [Line 6](#) reduces α by at most $\ell + 3p_{\max} \leq \text{poly}(\varepsilon^{-1}, p_{\max})$. This shows the claim. $\square$

As immediate consequence we get that REMOVE always computes a valid state $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{v})$ such that $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 + \|\Delta v\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max})$.

5.5 TECHNICAL DETAILS FOR ROBUST $Q||C_{\min}$

In this section, we give the full technical details necessary to prove that we can set up an ILP with parameters α , $\mu^{(\text{blue})}$ and ν at each step q that has an optimal solution $(x, \nu^{(\text{blue})})$, such that x is a $(1 - \mathcal{O}(\varepsilon))$ -competitive solution for the red machines, $\nu^{(\text{blue})}$ has $p^\top \nu^{(\text{blue})} \geq (1 + \varepsilon) \text{OPT}^\downarrow s^\top \mu^{(\text{blue})}$ for the blue machines, and $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 + \|\Delta\nu\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max})$ between any steps q and $q + 1$. We start with some redefinitions of concepts used in the proof of $Q||C_{\max}$; in fact, the statements proved in this section for the insertion/removal of jobs are almost always the “inverse” of their $Q||C_{\max}$ counterparts, i. e. follow the respective statements for removal/insertion of $Q||C_{\max}$ instead.

5.5.1 Some Preliminaries

Before we can formulate our procedure, we need to redefine some previously used definitions and statements to fit the new objective function. Recall that we assume that machine speeds are all powers of $(1 + \varepsilon)$. Here, we can achieve this property by rounding each machine speed down to the next power of $(1 + \varepsilon)$. For $Q||C_{\min}$, this process introduces an additional multiplicative error of at most $(1 + \varepsilon)^{-1}$ on the solution. We redefine OPT^* to map to the maximum minimum completion time over all schedules. Recall that the goal of dynamic $Q||C_{\min}$ is to find a schedule with maximum minimal completion time for every step q over the duration, where either $\nabla^{(q+1)}\nu = \nabla^{(q)}\nu + \mathfrak{e}^{(j)}$, for some $j \in [d]$, or $\nabla^{(q+1)}\nu = \nabla^{(q)}\nu - \mathfrak{e}^{(j)}$, for some $j \in [d]$ and $\nabla^{(q)}\nu_j > 0$. We slightly increase our threshold ℓ to fit our new objective.

$$\ell := (1 + \varepsilon)^{1 + \lceil \log_{1+\varepsilon} 3\varepsilon^{-1} p_{\max} \rceil}.$$

Note that

$$\varepsilon(1 + \varepsilon)^{-1}\ell \geq \varepsilon\ell \geq 3p_{\max}, \quad \text{and } \ell \leq \text{poly}(\varepsilon^{-1}, p_{\max}).$$

We use the following ILP, a version of [\(ILP^{dynamic}\)](#) with modified dual knapsack constraint and modified objective $\varphi^{(\text{lex})'}$. The objective function $\varphi^{(\text{lex})'}$ here encodes “Maximizing the minimum completion time” on the given set of configurations – this can simply be done by modifying the order $\sqsubseteq$ on the elements as defined in [Section 5.4](#) to order primarily by ascending, instead of descending, completion time.

(ILP^{dynamic}($C_{\min}$))

$$\min \varphi^{(\text{lex})'}(x)$$

s.t.

$$p^\top \nu^{(\text{blue})} \geq \alpha \tag{5.4}$$

$$\sum_{c \in \mathcal{C}(\ell)} x_{t,c} = \mu_t - \mu_t^{(\text{blue})} \quad \forall t \in [\tau] \quad (5.5)$$

$$\sum_{\substack{t \in [\tau] \\ c \in \mathcal{C}(\ell)}} x_{t,c} \cdot c + \nu^{(\text{blue})} = \nu \quad (5.6)$$

with variables

$$\text{var } x_{t,c} \in \mathbb{N}_{\geq 0} \quad \forall t \in [\tau], c \in \mathcal{C}(\ell)$$

$$\text{var } \nu^{(\text{blue})} \in \mathbb{N}_{\geq 0}^d$$

with parameters

$$\text{param } \nu \in \mathbb{N}_{\geq 0}^d$$

$$\text{param } \mu^{(\text{blue})} \in \mathbb{N}_{\geq 0}^\tau$$

$$\text{param } \alpha \in \mathbb{Q}_{\geq 0}$$

As our ILP again is parameterized by some value α , which now represents the part of the overall area on blue machines that *must* be used by the ILP, our next step is to formalize this interaction, i. e. ensure that we restrict ourselves to values of α that do not subceed the overall area of the blue machines. Here, we require the area of the blue machines to actually be a factor of $(1 + \varepsilon)$ higher than the current guess on the minimum completion time T .

Invariant 5.A (Compatible Coloring Invariant).

$$\alpha \geq (1 + \varepsilon) T s^\top \mu^{(\text{blue})} .$$

For fixed T and $\mu^{(\text{blue})}$, we will also call $(1 + \varepsilon) T s^\top \mu^{(\text{blue})}$ the “required blue area (for T and $\mu^{(\text{blue})}$)”. Given a (approximate) minimum completion time T , we require the red schedule to not subceed T , and the blue schedule not to subceed $(1 + \varepsilon)T$. This, again, will later allow us to bound the number of machine recolourings by some kind of witness argument. We use the definition of “states” as before, but redefine validity for a state.

Definition 15 (T -validity for $\mathbb{Q} \parallel C_{\min}$). The (unique) optimal solution $(x, \nu^{(\text{blue})})$ of $(\text{ILP}^{\text{dynamic}}(C_{\min}))$ for a state $(\alpha, \mu^{(\text{blue})}, \nu)$ is called the *associated solution*. The (ILP) *minimum completion time of a state* is the (ILP) minimum completion time of the associated solution, i. e. $C_{\min}(x)$. If the ILP is infeasible, we define the minimum completion time to be $-\infty$. A state S is T -valid iff Invariant 3.A and Invariant 5.A both hold for T ; and the associated solution of S has (ILP) minimum completion time at least T .

Note that if there is a T -valid state, there is also T' -valid state for any $T' \leq T$. We similarly define the Optimal ILP Minimum Completion Time, but redefine a state as valid if it is $\text{OPT}^\downarrow$ -valid.

Definition 16 (Optimal ILP Minimum Completion Time). Given a job vector $\nu \in \mathbb{N}_{\geq 0}^d$, $\text{OPT} \in \mathbb{Q}_{\geq 0}$ is the largest number such that there is a

OPT-valid state S . Moreover, we define $\text{OPT}^\downarrow$ to be the largest power of $(1 + \varepsilon)$ strictly smaller than OPT ; and $\text{OPT}^\uparrow := (1 + \varepsilon) \text{OPT}^\downarrow$.

Definition 17 (Valid States). A state S is valid if it is $\text{OPT}^\downarrow$ -valid.

Note that $(1 + \varepsilon)^{-1} \geq (1 - \varepsilon)$, for all $\varepsilon > 0$. By the definition of [Invariant 5.A](#), $\text{OPT} \geq \text{OPT}^*$ does not necessarily hold. Instead, $\text{OPT} \geq (1 - \varepsilon) \text{OPT}^*$ holds: By definition of OPT^* and [Invariant 5.A](#), there always exists a $(1 + \varepsilon)^{-1} \text{OPT}^*$ -valid state. Thus, $\text{OPT} \geq (1 + \varepsilon)^{-1} \text{OPT}^*$ by definition of OPT and the rounding of machine speeds. As computing the assignment of blue jobs to blue machines introduces an additional multiplicative error of at most $(1 - \varepsilon) \text{OPT}$, and $\text{OPT}^\downarrow \geq (1 + \varepsilon)^{-1} \text{OPT}$, computing a solution to the ILP for any valid state always returns a at most $(1 - \varepsilon)^3 \in (1 - \mathcal{O}(\varepsilon))$ -approximate solution. We redefine the freeness of states: Here, it measures how much space can be additionally allotted to α , without changing the minimum completion time of the associated solution.

Definition 18 (κ -free States). A valid state $(\alpha, \mu^{(\text{blue})}, \nu)$ is κ -free if $(\alpha + \kappa, \mu^{(\text{blue})}, \nu)$ is valid.

Machines of critical-type are defined as before, except that the shorthand i now stands for $i = \text{critical-type}(\text{OPT}^\downarrow)$. Similarly, machines of this type are the ones that might be necessary to be recolored during a sequence of steps where $\text{OPT}^\downarrow$ does not change. This again follows from the fact that, as soon as a machine of capacity $\geq \ell$ exists, the change on the minimum completion time is bounded. To see this, we need to show that OPT changes by a factor of at most $(1 + \varepsilon)$, which implies similar bounds for $\text{OPT}^\uparrow$ and $\text{OPT}^\downarrow$.

Proposition 14 (Bounding Minimum Completion Time Changes). Consider $\nu \in \mathbb{N}_{\geq 0}^d$ and $j \in [d]$. Let $f(\cdot) \in \{\text{OPT}(\cdot), \text{OPT}^\downarrow(\cdot), \text{OPT}^\uparrow(\cdot)\}$. If $s_{\max} \text{OPT}^\downarrow(\nu + \mathbf{e}^{(j)}) \geq \ell$, then $\frac{f(\nu + \mathbf{e}^{(j)})}{f(\nu)} \leq (1 + \varepsilon)$, and if $j \in \text{supp}(\nu)$ and $s_{\max} \text{OPT}^\downarrow(\nu) \geq \ell$, $\frac{f(\nu - \mathbf{e}^{(j)})}{f(\nu)} \geq (1 + \varepsilon)^{-1} \geq (1 - \varepsilon)$.

Proof. As [Proposition 3](#) also holds for $Q||C_{\min}$, we can prove the statement by simply exchanging the assumptions $s_{\max} \text{OPT}^\uparrow(\nu + \mathbf{e}^{(j)}) \geq \ell$ and $s_{\max} \text{OPT}^\uparrow(\nu) \geq \ell$ by $s_{\max} \text{OPT}^\downarrow(\nu + \mathbf{e}^{(j)}) \geq \ell$ and $s_{\max} \text{OPT}^\downarrow(\nu) \geq \ell$, respectively, in the proof of [Lemma 21](#). $\square$

This then implies that recolouring all machines of critical-type is enough to maintain [Invariant 3.A](#) and feasibility of the solution. We thus again adhere to [Invariant 4.A](#) during the design of our algorithm. We are now ready to describe how we want to modify a state each time we add or remove a job, such that the new state has distance bounded in $\text{poly}(\varepsilon^{-1}, p_{\max})$.

5.5.2 Handling Removal

For $Q||C_{\min}$, we begin with handling the removal of jobs. The change of parameters $\mu^{(\text{blue})}$ and α depends on how much we have to increase the κ -freeness of the current state to offset the removal of a job. We again make a general observation about solutions with enough κ -freeness.

Lemma 30 (Removing a job from a κ -free state). *Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ be valid and κ -free. Then for any $\nu^\Delta \in \mathbb{N}_{\geq 0}^d$ with $p^\top \nu^\Delta \leq \kappa$, $(\alpha, \mu^{(\text{blue})}, \nu - \nu^\Delta)$ is $\text{OPT}^\downarrow(\nu)$ -valid.*

Proof. Let $S' := (\alpha + p^\top \nu^\Delta, \mu^{(\text{blue})}, \nu)$. Then S' is valid by [Definition 18](#) and the fact that $p^\top \nu^\Delta \leq \kappa$. Let i be the machine where ν^Δ is to be removed.

We first assume i is a red machine. We now remove ν^Δ from i , decreasing i 's completion time by $p^\top \nu^\Delta$. We now decrease the required blue area from $\alpha + p^\top \nu^\Delta$ back to α . By this, we gain jobs of total processing time at least $\alpha + p^\top \nu^\Delta$ to be scheduled on the red machines. We schedule these jobs on i . Thus, i 's completion time is at least the same as before, i.e. at least $\text{OPT}^\downarrow$. Therefore, since the completion times of all other machines remain at least $\text{OPT}^\downarrow$, there exists a solution with minimum completion time at least $\text{OPT}^\downarrow$. Since the associated solution of this new state maximizes the minimum completion time, it has minimum completion time at least $\text{OPT}^\downarrow$ as well.

Now, assume i is a blue machine. We can simply remove ν^Δ from S' and decrease the required blue area from $\alpha + p^\top \nu^\Delta$ back to α to get a valid state. $\square$

So if a state is at least p_j -free, we can directly remove a job of type j from the ILP without changing any of the state parameters, and still stay valid. If this is not possible, we may have increase the freeness somehow. Luckily, when coloring a machine of critical-type($\text{OPT}^\downarrow(\nu)$) blue $\rightarrow$ red, we gain at least $p_{\max}$ freeness due to the definition of ℓ and the additional error allowed on blue machines by [Invariant 5.A](#), leading to the following lemma.

Lemma 31 (Recoloring a Blue Machine). *If $S := (\alpha, \mu^{(\text{blue})}, \nu)$ is valid and there is a blue machine of type $t \in [\tau]$ with $s_t \text{OPT}^\downarrow = \ell$, then $S' := (\alpha - ((1 + \varepsilon)s_t \text{OPT}^\downarrow - p_{\max}), \mu^{(\text{blue})} - \mathbb{e}^{(t)}, \nu)$ is valid and $p_{\max}$ -free. Moreover, $C_{\min}(S') \geq C_{\min}(S)$.*

Proof. Let i be the machine to be recolored and t its type. Consider an assignment ν^Δ of blue jobs to i such that each blue machine of type $t \in [\tau]$ is assigned jobs of total load at least $(1 + \varepsilon)s_t \text{OPT}^\downarrow(\nu) - p_{\max}$. Such an assignment can be found by a simple greedy algorithm (e.g. list-scheduling), as S is valid and thus [Invariant 5.A](#) holds for $T \leftarrow \text{OPT}^\downarrow(\nu)$. Then $(\alpha - ((1 + \varepsilon)s_t \text{OPT}^\downarrow - p_{\max}), \mu^{(\text{blue})} - \mathbb{e}^{(t)}, \nu)$ has a minimum completion time no smaller than $(\alpha, \mu^{(\text{blue})}, \nu)$, as we can

do the following modifications to the solution $(x, v^{(\text{blue})})$ associated to $(\alpha, \mu^{(\text{blue})}, \nu)$:

1. Set $x_{t, \nu^\Delta} \leftarrow x_{t, \nu^\Delta} + 1$, $\nu \leftarrow \nu + \nu^\Delta$ and $\mu_t \leftarrow \mu_t + 1$; then the modified solution is feasible and its minimum completion time did not decrease (as $\varepsilon \ell \geq 3p_{\max}$).
2. Set $\alpha \leftarrow \alpha - ((1 + \varepsilon)s_t \text{OPT}^\downarrow(\nu) - p_{\max})$, $\nu \leftarrow \nu - \nu^\Delta$, and $\nu^{(\text{blue})} \leftarrow \nu^{(\text{blue})} - \nu^\Delta$; then the job set is the same as in the beginning, the minimum completion time did not decrease and the dual knapsack constraint is not violated as $p^\top \nu^{(\text{blue})} \geq \alpha$, so $p^\top (\nu^{(\text{blue})} - \nu^\Delta) \geq \alpha - p^\top \nu^\Delta \geq \alpha - ((1 + \varepsilon)s_t \text{OPT}^\downarrow(\nu) - p_{\max})$.
3. $\mu_t^{(\text{blue})} \leftarrow \mu_t^{(\text{blue})} - 1$ and $\mu_t \leftarrow \mu_t - 1$, which is a change of parameters, but does not change the RHS of the ILP where only $\mu - \mu^{(\text{blue})}$ appears.

The resulting state is at least $p_{\max}$ -free, since the newly red machine i has completion time at least $(1 + \varepsilon)s_i \text{OPT}^\downarrow - p_{\max}$ (by simple greedy assignment), even though its minimum required completion time for validity as a red machine is only $s_i \text{OPT}^\downarrow$. Thus, increasing α by $p_{\max}$ leads to a valid state, since $\varepsilon s_i \text{OPT}^\downarrow - p_{\max} = \varepsilon \ell - p_{\max} \geq 3p_{\max} - p_{\max} = 2p_{\max}$ and therefore, there is a combination of jobs currently placed on i of total processing time at least $p_{\max}$ and at most $2p_{\max}$ that can be placed on blue machines without violating validity, implying $p_{\max}$ -freeness of the current state. This implies $C_{\min}(S') \geq C_{\min}(S)$ as well. $\square$

By [Invariant 4.A](#) we may only recolor machines of critical type blue $\rightarrow$ red to gain $p_{\max}$ -freeness. What if all machines of this type have already been recolored? We then can make the following observation about the validity and minimum completion time of such solutions.

Lemma 32. *Let ν be a job set and for all $t \in [\tau]$, $\mu_t^{(\text{blue})} = \mu_t$ if $s_t \text{OPT}^\downarrow > \ell$ and 0 otherwise. Then $\left(\left\lceil (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})} \right\rceil, \mu^{(\text{blue})}, \nu \right)$ is valid and its minimum completion time is at least OPT.*

Proof. At first glance it might seem like a contradiction that the minimum completion time could be more than OPT. But this actually can happen, as [Definition 16](#) differs from [Definition 17](#) in terms of the parameterization of the invariants. In [Definition 16](#), they have to hold for OPT, while for a valid solution they are only required to hold for $\text{OPT}^\downarrow$. Since our definition of minimum completion time of a state only considers $C_{\min}(x)$ of the associated solution $(x, \nu^{(\text{blue})})$, we can indeed have a minimum completion time above OPT. Let $\tilde{S} := (\tilde{\alpha}, \tilde{\mu}^{(\text{blue})}, \nu)$ be an OPT-valid state, i.e. such that the invariants [3.A](#) and [5.A](#) both hold for $T \leftarrow \text{OPT}$, and the associated solution has minimum completion

time at least OPT . Such a state always exists by definition of OPT . Then

$$\begin{aligned}\bar{\alpha} &\geq (1 + \varepsilon) \text{OPT}(\nu) s^\top \bar{\mu}^{(\text{blue})} \geq (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \bar{\mu}^{(\text{blue})} \\ &= (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})} + (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\bar{\mu}^{(\text{blue})} - \mu^{(\text{blue})}) .\end{aligned}$$

Note that $\mu^{(\text{blue})} \leq \bar{\mu}^{(\text{blue})}$, as otherwise, by assumption on the form of $\mu^{(\text{blue})}$, under $\bar{\mu}^{(\text{blue})}$ either less than μ_t machines are colored blue for some $t \in [\tau]$, or under μ_t a machine of some type $t \in [\tau]$ with $s_t \text{OPT}^\downarrow < \ell$ is colored blue, which would violate [Invariant 3.B](#). So in particular, by subtracting $(1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\mu^{(\text{blue})} - \bar{\mu}^{(\text{blue})})$ on each side of the inequality and rounding up, we have

$$\begin{aligned}\alpha' &:= \left\lceil \bar{\alpha} - (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\mu^{(\text{blue})} - \bar{\mu}^{(\text{blue})}) \right\rceil \\ &\geq \left\lceil (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})} \right\rceil .\end{aligned}$$

By inductively applying [Lemma 31](#) it follows that $C_{\min}(\alpha', \mu^{(\text{blue})}, \nu) \geq \text{OPT}$. Thus, as $\alpha' \geq \left\lceil (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top \mu^{(\text{blue})} \right\rceil$, also

$$C_{\min}(\left\lceil (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})} \right\rceil, \mu^{(\text{blue})}, \nu) \geq \text{OPT} .$$

□

We can use this statement now to show the validity of states where a job is removed and $\text{OPT}^\downarrow$ doesn't change.

Lemma 33 (Existence of a Blue Machine). *Consider $j \in [d]$. If $(\alpha, \mu^{(\text{blue})}, \nu)$ is valid and $\text{OPT}^\downarrow(\nu) = \text{OPT}^\downarrow(\nu - \mathbf{e}^{(j)})$, then either $(\alpha - \kappa, \mu^{(\text{blue})}, \nu - \mathbf{e}^{(j)})$ is valid for some $\kappa \in [p_j]_0$, or there is a blue machine of capacity ℓ .*

Proof. Assume that $(\alpha - \kappa, \mu^{(\text{blue})}, \nu)$ is invalid for all $\kappa \in [p_j]_0$. Then for some $\kappa \in [p_j - 1]_0$ we have $\alpha - \kappa = \left\lceil (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})} \right\rceil$, as otherwise, [Definition 18](#) together with [Lemma 30](#) lead to a contradiction for $\alpha \leftarrow \alpha - p_j$. If there was no blue machine of capacity ℓ , we would have colored all machines red that we are allowed to under [Invariant 3.A](#), and with $\alpha \leftarrow \alpha - \kappa$ we would have lowered the area as much as is allowed under [Invariant 5.A](#). This is a contradiction to [Lemma 32](#). □

To now prove correctness and bounded distance, we need the following proposition.

Proposition 15 (Decreasing α for valid states). *If $S = (\alpha, \mu^{(\text{blue})}, \nu)$ is valid, then $S' = (\alpha', \mu^{(\text{blue})}, \nu)$ is valid for any $\alpha' \in \left[(1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})}; \alpha \right]$.*

Proof. Let $(x, \nu^{(\text{blue})})$ be the solution associated to S . Clearly, $(x, \nu^{(\text{blue})})$ is a feasible solution to S' . Moreover, $C_{\min}(x) \geq \text{OPT}^\downarrow$ as S is valid, and since $\text{OPT}^\downarrow$ is the same for S' (as it only depends on ν), S' is valid. □

We are now ready to describe our algorithm for removing a job for a state S , such that the new job set is valid to the returned state, and the distance from S to the returned state is bounded. The algorithm is called **REMOVE'** and can be found in [Algorithm 13](#). It follows directly

Algorithm 13 Removing a job of type j , given the current state S

```

1: function REMOVE'(j, S)
2:   S is of the form  $(\alpha, \mu^{(\text{blue})}, \nu)$ 
3:   if  $\text{OPT}^\downarrow(\nu - \mathbb{E}^{(j)}) = \text{OPT}^\downarrow(\nu)$  then
4:     if S is not  $p_j$ -free then
5:        $\alpha \leftarrow \max \left\{ (1 + \varepsilon) \text{OPT}^\downarrow(\nu - \mathbb{E}^{(j)})_{S^\top \mu^{(\text{blue})}}, \alpha - p_j \right\}$ 
6:       if S is (still) not  $p_j$ -free then
7:          $t \leftarrow \text{critical-type}(\text{OPT}^\uparrow(\nu))$   $\triangleright$  Note that  $t \neq \perp$ 
8:          $\mu^{(\text{blue})} \leftarrow \mu^{(\text{blue})} - \mathbb{E}^{(t)}$   $\triangleright$  Note that  $\mu_t^{(\text{blue})} \neq \emptyset$ 
9:          $\alpha \leftarrow \alpha - (1 + \varepsilon)_{S_t} \text{OPT}^\downarrow(\nu)$ 
10:      end if
11:    end if
12:  else
13:    if  $(\alpha, \mu^{(\text{blue})}, \nu - \mathbb{E}^{(j)})$  is valid then
14:       $\alpha \leftarrow (1 + \varepsilon) \text{OPT}^\uparrow(\nu - \mathbb{E}^{(j)})_{S^\top \mu^{(\text{blue})}}$   $\triangleright$  Only needed for
      INSERT'
15:    else
16:       $\alpha \leftarrow \alpha - p_j$ 
17:    end if
18:  end if
19:  return  $(\alpha, \mu^{(\text{blue})}, \nu - \mathbb{E}^{(j)})$ 
20: end function

```

directly from the definition of the algorithm that **REMOVE'** does not violate [Invariant 4.A](#).

Also here, we rephrase the algorithm as a lemma, and directly show that it always computes a valid state, and that the resulting state has parameters whose difference to the parameters of the input state is bounded in $\text{poly}(p_{\max}) \leq \text{poly}(\varepsilon^{-1}, p_{\max})$. This will then significantly simplify the proofs of further properties.

Lemma 34 (REMOVE' Rephrased). *Given a valid state $S = (\alpha, \mu^{(\text{blue})}, \nu)$ and $j \in [d]$, the call **REMOVE'**(j, S)*

- (i) *if $(\alpha, \mu^{(\text{blue})}, \nu - \mathbb{E}^{(j)})$ is valid, returns $(\alpha, \mu^{(\text{blue})}, \nu - \mathbb{E}^{(j)})$ if $\text{OPT}^\downarrow(\nu) = \text{OPT}^\downarrow(\nu - \mathbb{E}^{(j)})$, and $((1 + \varepsilon) \text{OPT}^\uparrow(\nu - \mathbb{E}^{(j)})_{S^\top \mu^{(\text{blue})}}, \mu^{(\text{blue})}, \nu - \mathbb{E}^{(j)})$ else, where $(1 + \varepsilon) \text{OPT}^\uparrow(\nu - \mathbb{E}^{(j)})_{S^\top \mu^{(\text{blue})}} \geq \alpha - p_j$; otherwise*
- (ii) *returns a valid state $(\bar{\alpha}, \mu^{(\text{blue})}, \nu - \mathbb{E}^{(j)})$ with $\bar{\alpha} \in [\alpha - 2p_j, \alpha]$; or otherwise*

- (iii) correctly concludes that $(\bar{\alpha}, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$ is invalid for all $\bar{\alpha}$, and returns a valid state of the form $(\bar{\alpha}, \mu^{(\text{blue})} - \mathfrak{e}^{(\text{critical-type}(\text{OPT}^\uparrow(\nu)))}, \nu - \mathfrak{e}^{(j)})$ where $\bar{\alpha} > \alpha - (1 + \varepsilon)\ell - p_j$.

Proof. In the following, we can assume that there always exists a machine i with $s_i = s_{\max}$, and $s_{\max} \text{OPT}^\downarrow(\nu) \geq \ell$, as otherwise, it holds for both the input and return state that $\alpha = 0$, $\mu^{(\text{blue})} = \emptyset$ by [Invariant 3.A](#), and thus, (i) is always fulfilled.

Case 1: $(\alpha, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$ is valid.

Case 1.1: $\text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)}) = \text{OPT}^\downarrow(\nu)$. Then the if-branch of the first if-statement is entered and neither α nor $\mu^{(\text{blue})}$ are changed. Since the algorithm then returns $(\alpha, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$, this shows the claim, as (i) is satisfied.

Case 1.2: $\text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)}) < \text{OPT}^\downarrow(\nu)$. Then the else-branch of the first if statement is entered. Within this branch, the first if-branch of the next if-statement is entered due to the assumption of Case 1. Thus, the algorithm eventually returns

$$\left((1 + \varepsilon) \text{OPT}^\uparrow(\nu - \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})}, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)} \right).$$

By [Proposition 14](#) and using the fact that $\text{OPT}^\uparrow$ and $\text{OPT}^\downarrow$ are restricted to integer powers of $(1 + \varepsilon)$,

$$(1 + \varepsilon) \text{OPT}^\uparrow(\nu - \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} = (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})} \leq \alpha,$$

where the latter follows from the fact that the initial state was valid and thus satisfied [Invariant 5.A](#). Now, if $(1 + \varepsilon) \text{OPT}^\uparrow(\nu - \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} = (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})} < \alpha - p_j$, then the state $((1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})}, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$ would be valid for $\text{OPT}^\downarrow(\nu)$ by [Lemma 30](#), and thus $\text{OPT}^\downarrow(\nu) = \text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)})$, a contradiction to the assumption of Case 1.2. Thus, $(1 + \varepsilon) \text{OPT}^\uparrow(\nu - \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} \geq \alpha - p_j$. Since $\mu^{(\text{blue})}$ remains unchanged, this shows the claim, as (i) is satisfied.

Case 2: $(\alpha, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$ is invalid and $\text{OPT}^\uparrow(\nu - \mathfrak{e}^{(j)}) \neq \text{OPT}^\uparrow(\nu)$. Then only [Line 16](#) is executed. By definition of this line, α is decreased by at most p_j . The returned state is valid: By the assumption, we know that $\text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)}) < \text{OPT}^\downarrow(\nu)$ and that there exists a machine i with speed $s_{\max}$ such that $s_{\max} \text{OPT}(\nu) \geq \ell$. This machine i 's capacity is then decreased by at least $3p_{\max}$, as here, $\text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)}) = (1 + \varepsilon) \text{OPT}^\downarrow(\nu)$ by definition of $\text{OPT}^\downarrow$, and therefore, i 's capacity is decreased by at least $(1 + \varepsilon)^{-1} \varepsilon \ell \geq 3p_{\max}$, since i had capacity at least $(1 + \varepsilon)^{-1} \ell$ before by [Proposition 14](#).

Thus, if i is colored blue, [Invariant 5.A](#) is maintained. If i is colored red instead, j can be removed without changing α at all, and the state $(\alpha, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$ would be valid, a contradiction to the assumption of Case 2. It follows that (ii) is satisfied.

Case 3: $(\alpha, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$ is invalid and $\text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)}) = \text{OPT}^\downarrow(\nu)$. Then the second if-branch is entered because of [Lemma 30](#).

Case 3.1: there is a valid state of the form $(\tilde{\alpha}, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$. Let α' be the value of α after [Line 5](#). We prove that (ii) will always be satisfied.

Case 3.1.1: $\alpha' = \alpha - p_j$. Then $(\alpha', \mu^{(\text{blue})}, \nu)$ is p_j -free by [Definition 18](#) and hence $(\alpha', \mu^{(\text{blue})}, \nu)$ is valid by [Lemma 30](#). Moreover, $\tilde{\alpha} := \alpha' = \alpha - p_j$ satisfies the condition of (ii) on $\tilde{\alpha}$.

Case 3.1.2: $\alpha' = (1 + \varepsilon) \text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} > \alpha + p_j$. Then, with α' , we reduce α as much as is allowed under [Invariant 5.A](#). By our case (Case 3.1), we know that there is a valid state of the form $(\tilde{\alpha}, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$. Since α' is the smallest possible value (otherwise [Invariant 5.A](#) is violated), $\alpha' \leq \tilde{\alpha}$. By [Proposition 15](#), $(\alpha', \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$ is valid. Setting $\tilde{\alpha} := \alpha' < \alpha + p_j$ thus shows the claim.

Case 3.2: $(\tilde{\alpha}, \mu^{(\text{blue})}, \nu - \mathfrak{e}^{(j)})$ is invalid for all $\tilde{\alpha}$. Then the second if-statement is entered with usable blue area $\alpha' = (1 + \varepsilon) \text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)}) s^\top \mu^{(\text{blue})} > \alpha - p_j$ (otherwise, $(\alpha', \mu^{(\text{blue})}, \nu)$ was p_j -free by [Definition 18](#) $\nmid$). We show that (iii) is satisfied.

Case 3.2.1: $t \neq \perp \wedge \mu_t^{(\text{blue})} \neq \mathbb{O}$. Then $\tilde{\alpha} := \alpha' - (1 + \varepsilon)\ell$ and the state $(\tilde{\alpha}, \mu^{(\text{blue})} - \mathfrak{e}^{(t)}, \nu - \mathfrak{e}^{(j)})$ is returned by our algorithm, which is valid by [Lemma 30](#) as $(\tilde{\alpha}, \mu^{(\text{blue})} - \mathfrak{e}^{(t)}, \nu)$ is $p_{\max}$ -free.

Case 3.2.2: $t = \perp \vee \mu_t^{(\text{blue})} = \mathbb{O}$. Then REMOVE' would fail, as it tries to recolor a machine of undefined type or a machine of a type where all machines are red already. We show that this case cannot occur. First, we know that, since $\text{OPT}^\downarrow(\nu) = \text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)})$, in this case there is at least one blue machine by [Lemma 33](#). By [Invariant 4.A](#) this implies that at some point before, a job of type j' was removed which lead to a decrease of $\text{OPT}^\downarrow$ while there was still a blue machine with capacity $s_{j'} \text{OPT}^\downarrow = \ell$. This leads to a contradiction, as, if at that time this machine was recolored blue $\rightarrow$ red, then j' could have been removed without decreasing $\text{OPT}^\downarrow$ by [Lemma 31](#). $\square$

As an immediate consequence, since $(1 + \varepsilon)\ell \leq \text{poly}(\varepsilon^{-1}, p_{\max})$, we get that REMOVE' always computes a valid state $\tilde{S} = (\tilde{\alpha}, \tilde{\mu}^{(\text{blue})}, \tilde{\nu})$ such that $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 + \|\Delta\nu\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max})$.

5.5.3 Handling Insertion

We now incorporate the insertion of jobs into the algorithm. We make use of the subroutine TRYRTB ([Algorithm 14](#)) that simply tries to recolor up to k machines red $\rightarrow$ blue without violating [Invariant 5.A](#). We then propose algorithm INSERT', to be found in [Algorithm 15](#),

as a procedure to insert a job, using `TryRTB` as a subroutine. Let $\mu^{(\text{red})} := \mu - \mu^{(\text{blue})}$.

Algorithm 14 Recoloring subroutine

```

1: function TryRTB( $k, \alpha, \mu^{(\text{blue})}, \nu$ )
2:   if  $\mu^{(\text{red})} \neq \mathbf{0}$  then
3:      $t \leftarrow \text{critical-type}(\text{OPT}^\uparrow(\nu))$ 
4:     if  $t \neq \perp$  then
5:        $k \leftarrow \min\{k, \mu_t^{(\text{red})}\}$ 
6:        $\mu^{(\text{blue})} \leftarrow \mu^{(\text{blue})} + k \cdot \mathbb{e}^{(t)}$ 
7:        $\alpha \leftarrow \alpha + (1 + \varepsilon) \cdot k s_t \cdot \text{OPT}^\downarrow(\nu)$ 
8:     end if
9:   end if
10:  return  $(\alpha, \mu^{(\text{blue})}, \nu)$ 
11: end function

```

Algorithm 15 Inserting a job of type j , given the current state S

```

1: function INSERT'( $j, S$ )
2:    $S$  is of the form  $(\alpha, \mu^{(\text{blue})}, \nu)$ 
3:    $t \leftarrow \text{critical-type}(\text{OPT}^\downarrow(\nu + \mathbb{e}^{(j)}))$ 
4:   if  $\text{OPT}^\downarrow(\nu + \mathbb{e}^{(j)}) > \text{OPT}^\downarrow(\nu)$  then
5:      $S \leftarrow \text{TryRTB}(m, S)$   $\triangleright$  Note that this modifies  $\mu^{(\text{blue})}$ 
6:      $\alpha \leftarrow (1 + \varepsilon) \cdot \text{OPT}^\downarrow(\nu + \mathbb{e}^{(j)}) s^\top \mu^{(\text{blue})}$ 
7:   end if
8:    $\nu \leftarrow \nu + \mathbb{e}^{(j)}$ 
9:   if  $S$  is  $(\varepsilon\ell)$ -free  $\wedge t \neq \perp \wedge \mu_t^{(\text{red})} > 0$  then
10:     $S \leftarrow \text{TryRTB}(1, S)$ 
11:   end if
12:   if  $S$  is  $p_{\max}$ -free  $\wedge (t = \perp \vee \mu_t^{(\text{red})} = 0)$  then
13:      $\alpha \leftarrow \min \left\{ \alpha + p_{\max}, (1 + \varepsilon) \cdot \text{OPT}^\uparrow(\nu) \cdot s^\top \mu^{(\text{blue})} \right\}$ 
14:   end if
15:   return  $(\alpha, \mu^{(\text{blue})}, \nu)$ 
16: end function

```

We now want to show similar properties as in the removal case, e.g. prove that the state computed by the algorithm is valid for the new job vector, and the change of the parameters is bounded by $\text{poly}(\varepsilon^{-1}, p_{\max})$. The next lemma is almost the inverse of [Lemma 31](#), except that we require S to be $(\varepsilon\ell)$ -free instead of gaining $p_{\max}$ -freeness.

Lemma 35 (Recoloring a Red Machine). *If $S := (\alpha, \mu^{(\text{blue})}, \nu)$ is $(\varepsilon\ell)$ -free and $\mu_t^{(\text{red})} > 0$, where $t = \text{critical-type}(\text{OPT}^\downarrow(\nu))$, then $S' := (\alpha + (1 + \varepsilon)\ell, \mu^{(\text{blue})} + \mathbb{e}^{(t)}, \nu)$ is valid.*

Proof. Let i be the machine of type $t := \text{critical-type}(\text{OPT}^\downarrow(\nu))$ to be recolored. Note that $\ell = s_i \text{OPT}^\downarrow(\nu)$ by definition of $\text{critical-type}(\cdot)$.

Now, let $(x, v^{(\text{blue})})$ be the solution associated to S , and σ_i the assignment of jobs to i according to x . Also note that $p^\top \sigma_i \geq \ell$. By [Definition 18](#) we know that $\bar{S}$ is valid. Moreover, as the job set is the same as in S , the values for $\text{OPT}^\downarrow$ also coincide.

Consider the following modifications to $(x, v^{(\text{blue})})$ and S :

1. Set $v \leftarrow v - \sigma_i$, $v^{(\text{blue})} \leftarrow v^{(\text{blue})} + \sigma_i$
2. Set $\mu_i^{(\text{blue})} \leftarrow \mu_i^{(\text{blue})} + 1$, $\mu_i^{(\text{red})} \leftarrow \mu_i^{(\text{red})} - 1$, $\alpha \leftarrow \alpha + (1 + \varepsilon)\ell$

We call the resulting state $\bar{S}$. This state $\bar{S}$ is valid: α can be increased by $(1 + \varepsilon)\ell$ compared to the valid state S , since we not only increase the load on the blue machines $v^{(\text{blue})}$ by $\sigma_i \geq \ell$, but S was also $(\varepsilon\ell)$ -free before, i.e. α can already be increased by $\varepsilon\ell$ due to [Definition 18](#) without violating the dual knapsack constraint. [Invariant 3.A](#) cannot be violated as we only recolored a machine of the critical type i . [Invariant 3.B](#) is not violated as recolouring a machine red $\rightarrow$ blue increases the required total blue area by $(1 + \varepsilon)\ell$, but we also increased α accordingly. Thus, the claim holds. $\square$

With this result in hand, we can prove the first part: Our algorithm does indeed always compute a valid state.

Lemma 36 (Correctness of INSERT'). *Given a valid state, INSERT' always computes a valid state.*

Proof. First, we show that the state computed after [Line 8](#) is valid. Afterwards, we argue that the remaining lines will not change a valid state into an invalid one. Let j be the type of the job that is inserted, i.e. $\Delta v = e^{(j)}$. We make a case distinction on ΔOPT . If $\Delta \text{OPT}^\downarrow = 0$, then simply inserting the job already yields a valid state, which is exactly what the algorithm computes. If $\Delta \text{OPT}^\downarrow > 0$, [Line 5](#) ensures that [Invariant 3.A](#) is not violated. The line thereafter ensures that [Invariant 5.A](#) is not violated: $(1 + \varepsilon) \text{OPT}^\downarrow (v + e^{(j)}) s^\top \mu^{(\text{blue})}$ is exactly the blue area we are required to fill when all machines of critical-type $(\text{OPT}^\downarrow (v + e^{(j)}))$ are colored red, which is true for the new critical-type $(\text{OPT}^\downarrow (v + e^{(j)}))$, since machines of this type had capacity $< \ell$ for $\text{OPT}^\downarrow$ and were thus colored red by validity of the input state. For $\mu^{(\text{blue})} = \mathbb{0}$, the equality above also holds as both sides are 0. In consequence, setting α to the value above satisfies [Invariant 5.A](#) for $T \leftarrow \text{OPT}^\downarrow (v + e^{(j)})$. Validity of the state then eventually follows from [Lemma 32](#). Since TRYRTB only recolors machines of critical-type, [Invariant 4.A](#) holds as well.

Now, let us consider what happens after [Line 8](#). If a machine is recolored due to an execution of [Line 10](#), then S was $(\varepsilon\ell)$ -free before. The resulting state is valid also, due to [Lemma 35](#). If [Line 13](#) is executed, then α is changed by at most $p_{\max}$, but the state was $p_{\max}$ -free before, implying it is valid thereafter. $\square$

Now, in order to prove bounded migration of our proposed procedure, we first need to introduce two upper bounds on the value of α , and show that they hold for every step of our algorithm. If both upper bounds hold for our starting state, this property is implied if they hold for any execution of REMOVE' and INSERT' on a valid state as well. These upper bounds will help us to first prove bounded κ -freeness of most states computed by our algorithm, and by that, help us show that the “critical lines” [Line 5](#) and [Line 6](#) of INSERT' do not increase $\mu^{(\text{blue})}$ and α too much, respectively. This then will imply bounded migration. The first upper bound is [Invariant 5.B](#).

Invariant 5.B. $i \neq \perp \implies (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top(\mu^{(\text{blue})} + \mathbb{e}^{(i)}) > \alpha$.

Note that $\mu^{(\text{blue})} + \mathbb{e}^{(i)}$ can have an entry in i greater than the available number of machines μ_i of this type; in this case, for the sake of simplicity, we define $(1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top(\mu^{(\text{blue})} + \mathbb{e}^{(j)})$ to be ∞ . For the initial solution with no jobs, [Invariant 5.B](#) holds as $\text{OPT} = \text{OPT}^\downarrow = 0$, implying that there cannot be a blue machine, and hence $i = \perp$. As we only use procedures REMOVE' and INSERT' to compute new states, if both procedure maintain the invariant, the whole algorithm does so.

Proposition 16. REMOVE' maintains [Invariant 5.B](#)

Proof. Since the value for α is only decreased by REMOVE' , there are only two scenarios where the claim does not become trivial: A change of $\text{OPT}^\downarrow$ and a recolouring of a blue machine (both cannot occur at the same time due to [Lemma 31](#)). Let $S := (\alpha, \mu^{(\text{blue})}, \nu)$ be the state before removal of job of type j and $\bar{S} := (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ with $\bar{\nu} := \nu - \mathbb{e}^{(j)}$ the state afterwards.

Case 1: $\bar{\mu}^{(\text{blue})} \neq \mu^{(\text{blue})} \wedge \text{OPT}^\downarrow(\nu) = \text{OPT}^\downarrow(\bar{\nu})$. This means that (iii) of [Lemma 34](#) “is executed”. Then $(\alpha', \mu^{(\text{blue})}, \bar{\nu})$ is invalid for all α' , so in particular for all $\alpha' \geq (1 + \varepsilon) \text{OPT}^\downarrow(\bar{\nu}) s^\top \mu^{(\text{blue})}$ which are all possible values that do not violate [Invariant 5.A](#). That means that the invalidity stems from a too small minimum completion time of the associated solutions. Recoloring a machine blue $\rightarrow$ red without decreasing the α -value will only make the situation worse, i.e. cannot lead to a larger minimum completion time. In consequence, the state $(\bar{\alpha}, \mu^{(\text{blue})} + \mathbb{e}^{(i)}, \bar{\nu})$ returned from REMOVE' must satisfy $\bar{\alpha} < (1 + \varepsilon) \text{OPT}^\downarrow(\bar{\nu}) s^\top \mu^{(\text{blue})} = (1 + \varepsilon) \text{OPT}^\downarrow(\bar{\nu}) s^\top (\bar{\mu}^{(\text{blue})} + \mathbb{e}^{(i)})$.

Case 2: $\bar{\mu}^{(\text{blue})} = \mu^{(\text{blue})} \wedge \text{OPT}^\uparrow(\nu) \neq \text{OPT}^\uparrow(\bar{\nu})$. When using REMOVE' , $\text{OPT}^\uparrow(\nu) \neq \text{OPT}^\uparrow(\bar{\nu})$ implies $\text{OPT}^\uparrow(\nu) > \text{OPT}^\uparrow(\bar{\nu})$. Let $\bar{i} := \text{OPT}^\uparrow(\bar{\nu})$ be the (new) critical type. In particular, we have $\bar{i} \neq i$. Due to [Definition 14](#) and [Invariant 3.A](#), we either have $i = \perp$ or $\mu_i^{(\text{blue})} = \mu_{\bar{i}}$. In the first case, the premise of the implications of [Invariant 4.B](#) evaluates to “false”, and [Invariant 4.B](#) holds. In the latter case, $(1 + \varepsilon) \text{OPT}^\uparrow(\bar{\nu}) s^\top (\mu^{(\text{blue})} + \mathbb{e}^{(i)})$ evaluates to ∞ , and [Invariant 4.B](#) holds as well, for all possible α . $\square$

Proposition 17. TRYRTB maintains *Invariant 5.B*.

Proof. Consider a call TRYRTB($k, \alpha, \mu^{(\text{blue})}, \nu$). We may wlog. assume that $i = \text{critical-type}(\text{OPT}^\downarrow) \neq \perp$, otherwise the function simply returns the input state. Moreover, we may wlog. assume that $k \leq \mu_i^{(\text{red})}$ as internally k is modified accordingly (for any $k' > \mu_i^{(\text{red})}$ the function behaves identically). Now, assume that *Invariant 5.B* holds for the initial state, i.e.

$$\alpha < (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\mu^{(\text{blue})} + \mathbf{e}^{(i)}) ,$$

and let $(\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \nu)$ be the state returned by the function. Then

$$\bar{\alpha} = \alpha + k(1 + \varepsilon) s_i \text{OPT}^\downarrow = \alpha + k(1 + \varepsilon) \ell .$$

Moreover,

$$\begin{aligned} (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \bar{\mu}^{(\text{blue})} &= (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\mu^{(\text{blue})} + k \mathbf{e}^{(i)}) \\ &= (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})} + k(1 + \varepsilon) s_i \text{OPT}^\downarrow \\ &= (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top \mu^{(\text{blue})} + k(1 + \varepsilon) \ell . \end{aligned}$$

We can then show the claim, i.e. given that *Invariant 5.B* did hold initially prove that it holds afterwards.

$$\begin{aligned} \alpha &< (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\mu^{(\text{blue})} + \mathbf{e}^{(i)}) \\ \iff \alpha + k(1 + \varepsilon) \ell &< (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\mu^{(\text{blue})} + \mathbf{e}^{(i)}) + k(1 + \varepsilon) \ell \\ \iff \bar{\alpha} &< (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\mu^{(\text{blue})} + (k + 1) \mathbf{e}^{(i)}) \\ &= (1 + \varepsilon) \text{OPT}^\downarrow(\nu) s^\top (\bar{\mu}^{(\text{blue})} + \mathbf{e}^{(i)}) \end{aligned}$$

□

Proposition 18. INSERT' maintains *Invariant 5.B*

Proof. Since the value for α is only increased by INSERT', there are only two scenarios where the claim does not become trivial: A change of $\text{OPT}^\downarrow$ or a recolouring of a red machine. Let $S := (\alpha, \mu^{(\text{blue})}, \nu)$ be the state before insertion of job of type j and $\bar{S} := (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ with $\bar{\nu} := \nu + \mathbf{e}^{(j)}$ the state afterwards.

Case 1: $\bar{\mu}^{(\text{blue})} \neq \mu^{(\text{blue})} \wedge \text{OPT}^\downarrow(\nu) = \text{OPT}^\downarrow(\bar{\nu})$. The invariant can only be violated due to an increase of α . As in this case, [Line 5](#) and [Line 6](#) are not executed, this can only be due to [Line 10](#) or [Line 13](#). The execution of [Line 10](#) maintains the invariant by [Proposition 17](#). Let $i =: \text{critical-type}(\text{OPT}^\downarrow(\nu))$. [Line 13](#) is only executed if $i = \perp$ or $\mu_i^{(\text{red})} = 0$, the latter of which is equivalent to $\mu_i^{(\text{blue})} = \mu_i$; in both cases, the invariant trivially holds, as either its premise is false or the LHS of the inequality evaluates to ∞ .

Case 2: $(\bar{\mu}^{(\text{blue})} = \mu^{(\text{blue})} \vee \bar{\mu}^{(\text{blue})} \neq \mu^{(\text{blue})}) \wedge \text{OPT}^\downarrow(v) \neq \text{OPT}^\downarrow(\bar{v})$.

Then, [Line 5](#) and [Line 6](#) are executed. Again, the execution of [Line 5](#) maintains the invariant by [Proposition 17](#).

Let $\check{i} := \text{critical-type}(\text{OPT}^\downarrow(\bar{v}))$. Then, after [Line 6](#), if $\mu_{\check{i}}^{(\text{blue})} < \mu_{\check{i}}^\gamma$,

$$\tilde{\alpha} := (1 + \varepsilon) s^\top \text{OPT}^\downarrow(\bar{v}) \bar{\mu}^{(\text{blue})} < (1 + \varepsilon) \text{OPT}^\downarrow(\bar{v}) s^\top (\bar{\mu}^{(\text{blue})} + \mathbb{e}^{(\check{i})}) .$$

Otherwise, if $\mu_{\check{i}}^{(\text{blue})} = \mu_{\check{i}}^\gamma$ or $\check{i} = \perp$, the invariant trivially holds as argued above. Thus, [Invariant 5.B](#) always holds when [Line 6](#) is executed.

In the current case of $\text{OPT}^\downarrow(v) \neq \text{OPT}^\downarrow(\bar{v})$, $\bar{S}$ can not be $p_{\max}$ -free (nor $(\varepsilon\ell)$ -free), as otherwise, we could simply remove j again without a minimum completion time decrease, implying $\text{OPT}^\downarrow(v) = \text{OPT}^\downarrow(\bar{v})$, a contradiction. Thus, neither [Line 10](#) nor [Line 13](#) are executed, no more changes to $\tilde{\alpha}$ are made, and [Invariant 5.B](#) always holds. $\square$

Our second upper bound is [Invariant 5.C](#).

Invariant 5.C. $\delta := (1 + \varepsilon) \text{OPT}^\uparrow(v) s^\top (\mu^{(\text{blue})} + \mu_{\check{i}}^{(\text{blue})} \mathbb{e}^{(i)}) \geq \alpha$.

As $\alpha = \delta = 0$ initially, this upper bound already holds for our starting state. Luckily, showing that [REMOVE'](#) or [INSERT'](#) maintain [Invariant 5.C](#) is less involved.

Proposition 19. [REMOVE'](#) maintains [Invariant 5.C](#).

Proof. If $\text{OPT}^\downarrow(v - \mathbb{e}^{(j)}) = \text{OPT}^\downarrow(v)$, [REMOVE'](#) only decreases α , while δ does not change. Else if $\text{OPT}^\uparrow(v + \mathbb{e}^{(j)}) \neq \text{OPT}^\uparrow(v)$, α is set to $(1 + \varepsilon) \text{OPT}^\uparrow(v - \mathbb{e}^{(j)}) s^\top \mu^{(\text{blue})} \leq \delta$ if the input state is valid already, or α is decreased by at least p_j if it is not. In the latter case, $\alpha < (1 + \varepsilon) \text{OPT}^\uparrow(v - \mathbb{e}^{(j)}) s^\top \mu^{(\text{blue})} \leq \delta$ after the decrease, as otherwise, the resulting state would be valid for $\alpha = (1 + \varepsilon) \text{OPT}^\downarrow(v) s^\top \mu^{(\text{blue})} \leq \delta$ as well, a contradiction to the assumption that it was not. $\square$

Proposition 20. [INSERT'](#) maintains [Invariant 5.C](#).

Proof. If $\text{OPT}^\downarrow(v + \mathbb{e}^{(j)}) > \text{OPT}^\downarrow(v)$, α is set to at most δ in [Line 6](#), as all machines of critical-type($\text{OPT}^\downarrow(v)$) are colored red $\rightarrow$ blue by [Line 5](#), and all machines of the new critical-type($\text{OPT}^\downarrow(v + \mathbb{e}^{(j)})$) are still colored red in the resulting state by virtue that [Invariant 3.A](#) was maintained for the input state. While recolouring a blue machine in [Line 10](#), only the capacity of the recolored machine is added to α , maintaining $\alpha \leq \delta$ by maintaining [Invariant 5.B](#). If [Line 13](#) is executed, α is set to at most δ as well. $\square$

Using these bounds given to us by these two invariants, we can prove upper bounds for the κ -freeness of states after the computation of a new state. We first consider [REMOVE'](#).

Proposition 21. Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ be a valid state and κ maximal such that S is κ -free, and $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \nu + \mathfrak{e}^{(j)})$ be a $\bar{\kappa}$ -free state returned by $\text{REMOVE}'(j, S)$. Let $\bar{t} := \text{critical-type}(\text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)}))$ and $\bar{\delta} := (1 + \varepsilon) \text{OPT}^\uparrow(\nu - \mathfrak{e}^{(j)})_{S^\top}(\mu^{(\text{blue})} + \mu_i^{(\text{blue})} \mathfrak{e}^{(\bar{t})})$. Then

$$\bar{\alpha} < \alpha \wedge \bar{\alpha} < \bar{\delta} \implies \bar{\kappa} \leq (1 + \varepsilon)\ell + p_{\max} - 1 .$$

Proof. Consider [Lemma 34](#); read the 3 cases as statements of an algorithm. If the algorithm terminates after (i), either $\bar{\alpha} = \alpha$ or $\bar{\alpha} = (1 + \varepsilon) \text{OPT}^\downarrow(\nu - \mathfrak{e}^{(j)})_{S^\top} \mu^{(\text{blue})} = \bar{\delta}$, where the last equality follows from the fact that no machines of type $\bar{t}$ are colored red in this case, implying $\mu_{\bar{t}}^{(\text{blue})} = \mu_{\bar{t}}$, and the statement is always true as its LHS evaluates to false. If the algorithm terminates after (ii), then $\bar{\kappa} < p_j$, as we reduced α by at most $2p_j$, but immediately remove p_j total processing time from $p^\top \nu^{(\text{blue})}$. Now assume that it terminates after (iii). Then, by the inequality $\bar{\alpha} > (1 + \varepsilon)\ell - p_j$, the statement follows directly, as this case only occurs if S was less than p_j -free before, $p^\top \nu^{(\text{blue})}$ is reduced by at least p_j (reducing the freeness by this amount), and thus it must be less than $((1 + \varepsilon)\ell + p_j)$ -free after. $\square$

We show a similar statement for the slack of α after using INSERT' .

Proposition 22. Let S be a valid state and κ maximal such that S is κ -free, $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ be a $\bar{\kappa}$ -free state returned by $\text{INSERT}'(j, S)$. Let $\bar{t} := \text{critical-type}(\text{OPT}^\downarrow(\nu))$ and $\bar{\delta} := (1 + \varepsilon) \text{OPT}^\uparrow(\nu)_{S^\top}(\mu^{(\text{blue})} + \mu_i^{(\text{blue})} \mathfrak{e}^{(\bar{t})})$. Then

$$(i) \quad \bar{\alpha} < \bar{\delta} \implies \bar{\kappa} \leq \max \{ \kappa, \varepsilon \ell \} .$$

$$(ii) \quad \text{OPT}^\downarrow(\nu + \mathfrak{e}^{(j)}) > \text{OPT}^\downarrow(\nu) \implies \bar{\kappa} < p_{\max} .$$

Proof. (i) If $\bar{\alpha} < \bar{\delta}$, then $\text{OPT}^\downarrow(\nu) = \text{OPT}^\downarrow(\nu + \mathfrak{e}^{(j)})$, as otherwise, [Line 5](#) and [Line 6](#) would have been executed and set $\alpha = (1 + \varepsilon) \text{OPT}^\downarrow(\nu + \mathfrak{e}^{(j)})_{S^\top}(\mu^{(\text{blue})} + \mu_i^{(\text{blue})} \mathfrak{e}^{(\bar{t})}) = \bar{\delta}$, where the last equality follows from [Proposition 3](#). If $\bar{\kappa} < \varepsilon \ell$, the claim directly holds. Otherwise, $\bar{\kappa} \geq \varepsilon \ell$. We backtrack which lines have been executed within INSERT' : If $\bar{S}$ is $(\varepsilon \ell)$ -free even after INSERT' , either [Line 10](#) or [Line 13](#) have been executed during this call, as both lines can only reduce the freeness, and thus, the state before these lines was at least $(\varepsilon \ell)$ -free as well, and one of the conditions before the respective line is then always true. Since $\bar{\alpha} < \bar{\delta}$ by assumption, and $\bar{\delta} = (1 + \varepsilon) \text{OPT}^\uparrow(\nu + \mathfrak{e}^{(j)})_{S^\top} \mu^{(\text{blue})}$ if $\bar{t} = \perp$ or $\mu_{\bar{t}}^{(\text{blue})} = \mu_{\bar{t}}$, each of both lines reduce α , and thus freeness, by at least $p_{\max}$ if they are executed, so the state S' before these lines is at least $\bar{\kappa} + p_{\max}$ -free. Since S' is derived from S by simply inserting some job, we can conclude that S is at least $\bar{\kappa}$ -free, which shows the claim.

(ii) $\text{OPT}^\downarrow(\nu + \mathfrak{e}^{(j)}) > \text{OPT}^\downarrow(\nu)$ holds by assumption. If $\bar{S}$ was at least $p_{\max}$ -free, removing a job of type j again would yield a state S' with $\text{OPT}^\downarrow(\nu + \mathfrak{e}^{(j)}) = \text{OPT}^\downarrow(\nu)$ by [Lemma 30](#), a contradiction. $\square$

Consequently, we combine the two previous statements to upper bound the slack of all states computed by our algorithm using only REMOVE' and INSERT' to compute new states.

Proposition 23. *Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ be some valid state, and let $i := \text{critical-type}(\text{OPT}^\downarrow(\nu))$ and $\delta := (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top(\mu^{(\text{blue})} + \mu_i^{(\text{blue})} \mathbb{e}^{(i)})$.*

Let $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ be a state computed either by $\bar{S} = \text{REMOVE}'(j, S)$ or $\bar{S} = \text{INSERT}'(j, S)$. Let $\bar{i} := \text{critical-type}(\text{OPT}^\downarrow(\bar{\nu}))$ and $\bar{\delta} := (1 + \varepsilon) \text{OPT}^\uparrow(\bar{\nu}) s^\top(\bar{\mu}^{(\text{blue})} + \bar{\mu}_{\bar{i}}^{(\text{blue})} \mathbb{e}^{(\bar{i})})$. Then

$$\begin{aligned} (\alpha < \delta &\implies S \text{ is at most } ((1 + \varepsilon)\ell + p_{\max} - 1)\text{-free}) \\ \implies (\bar{\alpha} < \bar{\delta} &\implies \bar{S} \text{ is at most } ((1 + \varepsilon)\ell + p_{\max} - 1)\text{-free}) . \end{aligned}$$

Proof. In the following, we assume that the LHS of the implication $(\alpha < \delta \implies S \text{ is at most } ((1 + \varepsilon)\ell + p_{\max} - 1)\text{-free})$ as well as the LHS of the implication on the RHS $(\bar{\alpha} < \bar{\delta})$ both evaluate to true, as otherwise, the implication always evaluates to true. We make a case distinction regarding $\text{OPT}^\downarrow(\bar{\nu})$ in relation to $\text{OPT}^\downarrow(\nu)$.

Case 1: $\text{OPT}^\downarrow(\bar{\nu}) < \text{OPT}^\downarrow(\nu)$. This only occurs if $\bar{S} = \text{REMOVE}'(j, S)$. Then $\bar{\alpha} < \alpha$, as, if $\bar{\alpha}$ is not decreased, $\bar{\alpha} = \delta$ by [Line 14](#) of REMOVE' and the fact that all machines of $\bar{i}$ are colored blue in this case, a contradiction to the assumption that $\bar{\alpha} < \bar{\delta}$. Thus, $\bar{S}$ is at most $((1 + \varepsilon)\ell + p_{\max} - 1)$ -free by [Proposition 21](#).

Case 2: $\text{OPT}^\downarrow(\bar{\nu}) > \text{OPT}^\downarrow(\nu)$. This only occurs if $\bar{S} = \text{INSERT}'(j, S)$. Then $\bar{S}$ is at most $((1 + \varepsilon)\ell + p_{\max} - 1)$ -free by [Proposition 22, \(ii\)](#).

Case 3: $\text{OPT}^\downarrow(\bar{\nu}) = \text{OPT}^\downarrow(\nu)$.

Case 3.1: $\bar{S} = \text{REMOVE}'(j, S)$ It holds that $\bar{\alpha} \leq \alpha$ by definition of REMOVE'. If $\bar{\alpha} = \alpha$, then the statement is true: Either $\bar{\alpha} = \alpha = \delta$, which makes the statement always true, or $\bar{\alpha} = \alpha < \delta$, implying $((1 + \varepsilon)\ell + p_{\max} - 1)$ -freeness for S , and thus $\bar{S}$ is at most $((1 + \varepsilon)\ell + p_{\max} - 1)$ -free since removing a job without decreasing α or $\text{OPT}^\downarrow$ can only decrease the freeness.

Else if $\bar{\alpha} < \alpha$, the statement is also true, as $\alpha \leq \delta$ by [Invariant 5.C](#) and $\delta = \bar{\delta}$ when $\text{OPT}^\downarrow(\bar{\nu}) = \text{OPT}^\downarrow(\nu)$, and thus $\bar{S}$ is at most $((1 + \varepsilon)\ell + p_{\max} - 1)$ -free by [Proposition 21](#).

Case 3.2: $\bar{S} = \text{INSERT}'(j, S)$ It holds that $\bar{\alpha} \geq \alpha$ by definition of INSERT'. Also, note that $\delta = \bar{\delta}$ when $\text{OPT}^\downarrow(\bar{\nu}) = \text{OPT}^\downarrow(\nu)$. Thus, $\bar{\alpha} \leq \bar{\delta} = \delta$, where the first inequality follows from [Invariant 5.C](#). If $\bar{\alpha} = \bar{\delta}$, the statement is always true. Else if $\alpha \geq \bar{\alpha} > \bar{\delta} = \delta$, S is at most $((1 + \varepsilon)\ell + p_{\max} - 1)$ -free by our assumptions, and [Proposition 22, \(i\)](#) then shows that $\bar{S}$ is at most $((1 + \varepsilon)\ell + p_{\max} - 1)$ -free as well, proving the statement.

This proves the claim. □

Since $(\alpha < \delta \implies S \text{ is at most } ((1 + \varepsilon)\ell + p_{\max} - 1)\text{-free})$ is true for our starting state ($\nu = 0$ and the state is 0-free), and we use only REMOVE' and INSERT' to compute new states, [Proposition 23](#) holds for every step of our algorithm. This implies the following corollary.

Corollary 3. *For each state $S = (\alpha, \mu^{(\text{blue})}, \nu)$ computed by our algorithm*

$$\alpha < \delta \implies S \text{ is at most } ((1 + \varepsilon)\ell + p_{\max} - 1)\text{-free}$$

is true, with $\dot{t} := \text{critical-type}(\text{OPT}^\downarrow(\nu))$ and $\delta = (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} + \mu_{\dot{t}}^{(\text{blue})} \oplus^{(\dot{t})})$.

We can use this bound on the κ -freeness of states to show the following claim on the coloring and α change caused by [Line 5](#) and [Line 6](#), respectively.

Proposition 24. *For any execution of INSERT'(j, S) on a state S produced by our algorithm and a of type j the following holds:*

- (i) *On any execution of [Line 5](#), at most $\frac{(1+\varepsilon)\ell + p_{\max}}{\varepsilon\ell}$ machines are recolored red $\rightarrow$ blue.*
- (ii) *On any execution of [Line 6](#), α is increased by at most $(1 + \varepsilon)\ell + 2p_{\max}$.*

Proof. Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ and $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \nu) = \text{INSERT}'(j, S)$. Also, let $\dot{t} = \text{critical-type}(\text{OPT}^\downarrow(\nu))$ and $\delta := (1 + \varepsilon) \text{OPT}^\uparrow(\nu) s^\top (\mu^{(\text{blue})} + \mu_{\dot{t}}^{(\text{blue})} \oplus^{(\dot{t})})$. [Line 5](#) and [Line 6](#) are only executed during INSERT'(j, S) if $\text{OPT}^\downarrow(\nu + \oplus^{(j)}) > \text{OPT}^\downarrow(\nu)$. Note that, in this case, $\delta = (1 + \varepsilon) \text{OPT}^\downarrow(\nu + \oplus^{(j)}) s^\top (\mu^{(\text{blue})} + \mu_{\dot{t}}^{(\text{blue})} \oplus^{(\dot{t})})$ by [Proposition 14](#). Thus, $\bar{\alpha} = \delta$ after execution of [Line 6](#).

$\alpha < \delta$ Then S is at most $((1 + \varepsilon)\ell + p_{\max} - 1)$ -free by [Corollary 3](#). Removing a job of type j again with REMOVE'($\bar{S}, j$) after its insertion yields a valid state $S' = (\alpha', \mu^{(\text{blue})'}, \nu)$ with $\text{OPT}^\downarrow(\nu) < \text{OPT}^\downarrow(\nu + \oplus^{(j)})$. For S' , $\mu_{\dot{t}}^{(\text{blue})'} \geq \mu_{\dot{t}} - 1$ and $\alpha' \geq \delta - p_j$, as, for $\text{OPT}^\downarrow(\nu) < \text{OPT}^\downarrow(\nu + \oplus^{(j)})$, REMOVE'($\bar{S}, j$) sets α first to δ , and then decreases α by at most p_j after.

(i). Assume that [Line 5](#) is executed on state S and job of type j and more than $\frac{(1+\varepsilon)\ell + p_{\max}}{\varepsilon\ell}$ machines are recolored red $\rightarrow$ blue. For this to occur, $\mu_{\dot{t}}^{(\text{red})} \geq \frac{(1+\varepsilon)\ell + p_{\max} + 1}{\varepsilon\ell}$ must hold at this point.

Then, since S' is at least 0-free due to its validity, S is at least $((1 + \varepsilon)\ell + p_{\max})$ -free, as each of the $\frac{(1+\varepsilon)\ell + p_{\max} + 1}{\varepsilon\ell}$ machines that are colored red in S but not in S' subtract at least $\varepsilon\ell$ from α , due to the maintaining of [Invariant 5.B](#). This is a contradiction to S is at most $((1 + \varepsilon)\ell + p_{\max} - 1)$ -free if $\alpha < \delta$ by [Corollary 3](#).

(ii). If $\alpha < \bar{\alpha} + (1 + \varepsilon)\ell + 2p_{\max} = \delta + (1 + \varepsilon)\ell + 2p_{\max}$, then S is at least $((1 + \varepsilon)\ell + p_{\max})$ -free for $\mu_i^{(\text{blue})} = \mu_i$, as $\alpha' \geq \delta + p_{\max}$ for the state S' with the same job set ν . This then holds for all $\mu_i^{(\text{blue})} \leq \mu_i$, as any additional red machine increases the κ -freeness by at least $\varepsilon\ell$ while maintaining [Invariant 5.B](#). This again is a contradiction to S is at most $((1 + \varepsilon)\ell + p_{\max} - 1)$ -free if $\alpha < \delta$, and thus, $\alpha \geq \delta + (1 + \varepsilon)\ell + 2p_{\max}$, and the statement follows as $\bar{\alpha} = \delta$.

$\alpha = \delta$ (i) and (ii) are both true in this case, as $\alpha = \bar{\alpha}$, and recolouring one machine red $\rightarrow$ blue with TRYRTB increases α by at least $(1 + \varepsilon)\ell$, implying that no machine can be recolored in the case of $\alpha = \bar{\alpha}$.

$\alpha > \delta$ can never occur by [Invariant 5.C](#). $\square$

We can now finally use this result to show that INSERT' has indeed bounded migration.

Lemma 37 (INSERT' Has Bounded Migration). *Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ be a valid state and $j \in \text{supp}(\nu)$. Then for $\bar{S} := (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu}) = \text{INSERT}'(j, S)$, $1 \leq |\Delta\alpha| + \|\Delta\nu\|_1 + \|\Delta\mu^{(\text{blue})}\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max})$.*

Proof. $\|\Delta\nu\|_1 = 1$ by definition of the problem. Due to [Proposition 24](#) (i), and the fact that in [Line 10](#), the only other red $\rightarrow$ blue recolouring, only 1 machine is recolored,

$$\|\Delta\mu^{(\text{blue})}\|_1 \leq \frac{(1 + \varepsilon)\ell + p_{\max}}{\varepsilon\ell} + 1 \leq \text{poly}(\varepsilon^{-1}, p_{\max}).$$

Thus, α is increased at most by

$$\text{poly}(\varepsilon^{-1}, p_{\max}) \cdot (1 + \varepsilon)\ell \leq \text{poly}(\varepsilon^{-1}, p_{\max})$$

by recolouring.

The only other times when α might be changed are [Line 13](#), where α is increased by at most $p_{\max} \leq \text{poly}(\varepsilon^{-1}, p_{\max})$, and [Line 6](#). By [Proposition 24](#) (ii), the execution of [Line 6](#) increases α by at most $(1 + \varepsilon)\ell + 2p_{\max} \leq \text{poly}(\varepsilon^{-1}, p_{\max})$. This shows the claim. $\square$

As immediate consequence we get that INSERT' always computes a valid state $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ such that $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 + \|\Delta\nu\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max})$.

5.6 PREPROCESSING AND ROUNDING AN INSTANCE

In this section, we describe the modifications necessary to approximate the optimal value of a problem instance without use of parameterization. To this end, we bound the parameters $p_{\max}$ and d in terms of ε^{-1} by preprocessing and rounding the instance. This allows us to formulate an ILP at each step q that approximates the instance with a factor of $(1 + \mathcal{O}(\varepsilon))$ for $Q||C_{\max}$, such that it holds for the RHS b at step q ,

and the RHS b' at step $q + 1$ of these ILPs that $\|b - b'\| \leq \text{poly}(\varepsilon^{-1})$. We then describe the necessary modifications to achieve the same result with an $(1 - \mathcal{O}(\varepsilon))$ approximation guarantee for $Q||C_{\min}$. Recall that we assume wlog. that for any occurring $\varepsilon > 0$, ε^{-1} is integral. We start by partitioning the job set into small jobs and large jobs: A job is large iff its processing time is at least $\varepsilon p_{\max}$, and small otherwise. On identical machines, a standard approach to deal with small jobs is to replace them by jobs of size $\varepsilon p_{\max}$ such that the area of the new jobs, i. e. the sum of their processing times, just cover the area of the small jobs. This causes an error of $< \varepsilon p_{\max}$ per machine, which is $\leq \text{OPT}^*$ due to $\text{OPT}^* \geq p_{\max}$. For uniform machines, this process is more involved, which leads to an additional load error of $\mathcal{O}(\varepsilon p_{\max})$ that cannot generally be estimated in terms of OPT^* . We discuss the details of this process in [Section 5.7](#) when proving [Theorem 17](#). For now, it suffices to know that we can deal with small jobs if we can deal with large jobs – the rough idea is that we group small jobs to chunks of size $\varepsilon p_{\max}$ that we represent using place-holder jobs which we refer to as *frames*. We thus start by only considering instances without small jobs.

We adhere to the rounding scheme introduced by Berndt et al. [[18](#)], and first describe the approach used to approximate the makespan $C_{\max}$. First, the interval of all possible processing times $[\varepsilon p_{\max}, p_{\max}]$ is split into $\log(\varepsilon^{-1})$ growing intervals of size $2^i \varepsilon p_{\max}$, for each $i = [0, \log(\varepsilon^{-1})]$. Each such interval I_i is then split again into $\lceil \varepsilon^{-1} \rceil$ subintervals $I_{i,k} = (b_{i,k}, b_{i,k+1}]$ of equal size, with $b_{i,k} = 2^i \varepsilon p_{\max} + k \varepsilon^2 2^i p_{\max}$ and $k \in [0, \lceil \varepsilon^{-1} - 1 \rceil]$. Each processing time is then rounded down to the next lower boundary $b_{i,k}$. As this is the same rounding scheme as used by Berndt et al. [[18](#)], with only the upper and lower bounds on the processing times $(1 - 2\varepsilon)T$ and εT being replaced by $p_{\max}$ and $\varepsilon p_{\max}$, the following statement still holds with only slight modification:

- Lemma 38** (Berndt et al. [[18](#)], Lemma 6.2). *1. The number of different rounded processing times is at most $\lceil \varepsilon^{-1} \rceil \lceil \log(\varepsilon^{-1}) + 1 \rceil$.*
- 2. A schedule $\tilde{\sigma}$ of the rounded processing times implies a schedule σ of the original processing time with $C_{\max}(\sigma) \leq (1 + \varepsilon)C_{\max}(\tilde{\sigma})$.*
- 3. We have $b_{i,k_1} + b_{i,k_2} = b_{i+1,(k_1+k_2)/2}$ for all i and k_1, k_2 with $k_1 \bmod 2 = k_2 \bmod 2$.*

After the rounding, we scale each of the resulting processing times by $\varepsilon^{-1} p_{\max}^{-1}$. As each rounded processing time corresponds to some boundary $b_{i,k}$, and all boundaries are multiples of $\varepsilon p_{\max}$, this scaling does not change any of the properties above. After the scaling, each rounded processing times is in $[1; \varepsilon^{-1}]$, but not necessarily integral any more. We again assume that machine speeds are all powers of $(1 + \varepsilon)$ by rounding them up to the next power of $(1 + \varepsilon)$, and let τ denote the number of different machine speeds after rounding, with

$s_{\max}$ being the largest speed after rounding and $s_{\min}$ being the smallest. In contrast d, τ is not bounded in terms of ε^{-1} after rounding.

After this procedure, it holds for the largest possible rounded processing time $p_{\max} = \varepsilon^{-1}$, and for the smallest rounded processing time $p_{\min} = 1$. Let here $p \in [p_{\min}; p_{\max}]^d$ be the vector of rounded job processing times and $\nu \in \mathbb{N}_{\geq 0}^d$ the corresponding multiplicity vector, i. e. for $j \in [d]$ there are ν_j jobs of rounded processing time p_j in the instance. Analogously, as before, let $s \in \mathbb{R}^\tau$ be the vector of machine speeds and $\mu \in \mathbb{N}_{\geq 0}^\tau$ the corresponding multiplicity vector. If not stated otherwise, we always assume that the instances we deal with in the upcoming section are instances after the application of this pre-processing procedure and are encoded by the vectors described above. Note that this procedure (for large jobs) induces a multiplicative error of at most $(1 + \varepsilon)^2 \in (1 + \mathcal{O}(\varepsilon))$, as each rounding step induces a multiplicative error of at most $(1 + \varepsilon)$. Exchanging $(1 + \varepsilon)$ by $(1 - 2\varepsilon)$ in all previous steps, rounding up jobs instead of down and machine speeds down instead of up, and replacing $\leq$ by $\geq$ in Lemma 38, 2, the procedure has same properties with an induced multiplicative error of at most $(1 - \varepsilon)^2 \in (1 - \mathcal{O}(\varepsilon))$ for approximating the “maximize the minimum completion”-objective.

Now, with this rounding in hand, jobs with processing time $b_{i,k}$ and $b_{i,k'}$ with $k \bmod k' = 2$ can be portrayed as a single job of processing time $b_{i+1, (k_1+k_2)/2} = b_{i,k} + b_{i,k'}$ according to Lemma 38, 3. This can now be used to replace configurations that contain more than 1 job of the same parity in the same interval i by configurations with a single job in the interval $i + 1$ representing 2 such jobs. To correctly count the number of jobs distributed by such configurations for some given job set in an ILP formulation, a set $\hat{\mathcal{C}}$ of auxiliary configurations is introduced. For i', k'_1, k'_2 with $k'_1 \bmod 2 = k'_2 \bmod 2$, let

- $\hat{c} = \hat{c}(i', k'_1, k'_2) \in \hat{\mathcal{C}}$ with $\hat{c}[i, k] = 2$ if $i = i'$ and $k = k'_1 = k'_2$
- $\hat{c}[i, k] = 1$ if $i = i'$ and $k \in \{k'_1, k'_2\}$ and $k'_1 \neq k'_2$
- $\hat{c}[i, k] = -1$ if $i = i' + 1$ and $k = (k'_1 + k'_2)/2$

Else, let $\hat{c}[i, k] = 0$. This implies that $|\hat{\mathcal{C}}| \leq \varepsilon^{-2} \log(\varepsilon^{-1})$. By definition, $\|\hat{c}\|_1 \leq 3$, for all $\hat{c} \in \hat{\mathcal{C}}$.

Let $\mathcal{C}(\ell)$ be the set of configurations after rounding, for scaled $\ell \leq \text{poly}(\varepsilon^{-1})$. Note that, for any $c \in \mathcal{C}(\ell)$ with $c[i, k_1] = 1$ and $c[i, k_2] = 1$, for some i, k_1, k_2 with $k_1 \bmod 2 = k_2 \bmod 2$, there exists some configurations $c' \in \mathcal{C}(\ell)$ and $\hat{c} \in \hat{\mathcal{C}}$, such that c' has the same entries as c , except $c'[i, k_1] = c'[i, k_2] = 0$ and $c'[i + 1, (k_1 + k_2)/2] = c[i + 1, (k_1 + k_2)/2] + 1$, such that $c = c' + \hat{c}$, i. e. c can be replaced by a combination of c' and $\hat{c}$. Now, let $\mathcal{C}_{\text{decr}}(u) = \{c \in \mathcal{C}(u) : \|c\|_1 \leq 2 \log(\varepsilon^{-1})\}$. As Berndt et al. [18] have shown, one can transform a vector that encodes a schedule using only configurations $c \in \mathcal{C}(u)$ to a vector that encodes the same schedule using only configurations

$c \in \mathcal{C}_{decr}(u) \cup \hat{\mathcal{C}}$ in linear time, and vice-versa [18, Lemma 6.3]. Note that $\|c\|_1 \leq \Theta(\log(\varepsilon^{-1}))$, for all $c \in \mathcal{C}_{decr}(u) \cup \hat{\mathcal{C}}$, and thus, $|\mathcal{C}_{decr}(u) \cup \hat{\mathcal{C}}| \leq \log(\varepsilon^{-1})^{\varepsilon^{-1} \log(\varepsilon^{-1})} + \varepsilon^{-2} \log(\varepsilon^{-1})$. We now use this configuration set in a modified (ILP^{dynamic}_{decr}) to approximate dynamic $Q||C_{\max}$.

$$\begin{aligned}
 & \text{(ILP}^{\text{dynamic}}_{\text{decr}}) \\
 & \min \varphi^{(\text{lex})}(x_{decr}) \\
 & \text{s.t.} \\
 & p^\top v^{(\text{blue})} \leq \alpha \\
 & \sum_{c \in \mathcal{C}_{decr}(\ell)} x_{t,c} = \mu_t - \mu_t^{(\text{blue})} \quad \forall t \in [\tau] \\
 & \sum_{\substack{t \in [\tau] \\ c \in \mathcal{C}_{decr}(\ell) \cup \hat{\mathcal{C}}}} x_{t,c} \cdot c + v^{(\text{blue})} = v \\
 & \text{with variables} \\
 & \text{var } x_{t,c} \in \mathbb{N}_{\geq 0} \quad \forall t \in [\tau], c \in \mathcal{C}_{decr}(\ell) \cup \hat{\mathcal{C}} \\
 & \text{var } v^{(\text{blue})} \in \mathbb{N}_{\geq 0}^d \\
 & \text{with parameters} \\
 & \text{param } v \in \mathbb{N}_{\geq 0}^d \\
 & \text{param } \mu^{(\text{blue})} \in \mathbb{N}_{\geq 0}^\tau \\
 & \text{param } \alpha \in \mathbb{Q}_{\geq 0}
 \end{aligned}$$

Let x_{decr} be the vector of entries of x of the form $x_{t,c}$, for $t \in [\tau]$ and $c \in \mathcal{C}_{decr}(\ell)$. As $\mathcal{C}_{decr} \subseteq \mathcal{C}$, applying $\varphi^{(\text{lex})}$ to x_{decr} still encodes “Minimizing the makespan” by the unique order of $\varphi^{(\text{lex})}$ on $\mathcal{C}$.

As argued above, schedules encoded by configurations in $\mathcal{C}_{decr}(\ell) \cup \hat{\mathcal{C}}$ can be transformed to schedules encoded by configurations in $\mathcal{C}(\ell)$ and vice-versa in linear time [18, Lemma 6.3]. This specifically implies that such transformed schedules have the same completion times on all machines, i. e. we can transform a lexicographically minimal solution (with minimal maximum completion time) of (ILP^{dynamic}_{decr}) to a schedule with configurations in $\mathcal{C}(\ell)$ of minimal maximum completion time. Now, in (ILP^{dynamic}_{decr}), the maximum absolute entry in every constraint besides the knapsack constraint

$$p^\top v^{(\text{blue})} \leq \alpha$$

is $\leq \Theta(\log(\varepsilon^{-1}))$, and every coefficient besides those in this constraint is integral. We first make the coefficients of this constraint integral by using a different rounding scheme on the job processing in $[\varepsilon p_{\max} \dots p_{\max}]$ times here. First, every jobs original processing time is rounded down to the value of the form $(1 + \varepsilon)^i \cdot \varepsilon p_{\max}$ for $i \in \mathbb{N}_{\geq 0}$. Then, each processing time is rounded down again to the next integer multiple of $\varepsilon p_{\max}$. Eventually, we scale each processing time by $\varepsilon^{-1} p_{\max}^{-1}$. Thereafter, every such rounded and scaled processing time

is in $[1.. \varepsilon^{-1}]$ and integral. This procedure induces a multiplicative error of at most $(1 + \varepsilon)^3 \in (1 + \mathcal{O}(\varepsilon))$, as each rounding step induces a multiplicative error of at most $(1 + \varepsilon)$. Moreover, there are again at most $d \in \mathcal{O}(\varepsilon^{-1} \cdot \log(\varepsilon^{-1}))$ possible processing times here. To bring the total maximum absolute entry down to $\log(\varepsilon^{-1})$, we use a technique introduced by Knop, Pilipczuk and Wrochna [91] called “Coefficient Reduction”.

Lemma 39 (Knop, Pilipczuk and Wrochna [91], Lemma 8). *Consider an instance $\{Ax = b, x \geq 0\}$ of ILP FEASIBILITY, where $b \in \mathbb{N}_{\geq 0}^k$ and A is a non-negative integer matrix with k rows and g columns. In polynomial time, this instance can be reduced to an equivalent instance $\{A'x = b', x \geq 0\}$ of ILP FEASIBILITY where A' is a $\{0, 1\}$ -matrix with $k' = \mathcal{O}(k \log(\|A\|_\infty))$ rows and $g' = g + \mathcal{O}(k \log(\|A\|_\infty))$ columns, and $b' \in \mathbb{N}_{\geq 0}^{k'}$ is a vector with $\|b'\|_\infty = \mathcal{O}(\|b\|_\infty)$.*

Here, ILP FEASIBILITY is defined as:

ILP FEASIBILITY

Instance: An ILP of the form $\{Ax = b, x \geq 0\}$, where A is an integer matrix with k rows and g columns.

Question: Is there an $x \in \mathbb{N}_{\geq 0}^g$ that satisfies $Ax = b$?

The submatrix as given by the constraint $p^\top \nu^{(\text{blue})} \leq \alpha$ of our $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$ is such an instance, with $A = p^\top$, $k = 1$, $g \in \mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}))$, $x = \nu^{(\text{blue})}$ and $b = \alpha$. Here, $\|p^\top\|_\infty \leq \varepsilon^{-1}$. We directly apply Lemma 39 on this constraint to replace it by $\Theta(\log(\varepsilon^{-1}))$ constraints with coefficients in $\{0, 1\}$ in $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$, introducing an additional $\log(\varepsilon^{-1})$ columns in the process. For the new RHS α' , $\|\alpha'\|_\infty = \mathcal{O}(\|\alpha\|_\infty)$ holds. Note that this α' may be not integral, but we deal with this while solving the ILP in Section 5.9. As the vector of variables $\nu^{(\text{blue})}$ itself is not changed by this replacement, it does change neither the feasibility nor optimality of solutions for $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$. The largest coefficient, and by extension the ℓ_1 -norm of any column, in the resulting ILP is now bounded by $\log(\varepsilon^{-1})$.

Let b be the RHS of the such further reduced $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$ at step q , b' the RHS at step $q + 1$. As INSERT and REMOVE are only dependent on the state parameters α , $\mu^{(\text{blue})}$ and ν , which are either not affected by the replacement of the configuration set and the replacement of constraints at all, or increased only by a constant factor in the case of α , these two functions can be used as before to go from step q to $q + 1$ such that $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 + \|\Delta\nu\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max}) \leq \text{poly}(\varepsilon^{-1})$, where the last inequality follows from the rounding and scaling applied in this section. An equivalent formulation with the same properties can be achieved for approximating dynamic $Q||C_{\min}$ by rounding down instead of up in each rounding step, changing “ $\leq$ ” to “ $\geq$ ” in the knapsack constraint, and using INSERT' and REMOVE' to

go from step q to $q + 1$, respectively. The rest then follows equivalently, and we get solutions with an approximation guarantee of $(1 - \mathcal{O}(\varepsilon))$.

5.7 A (DYNAMIC) GROUPING TECHNIQUE

In this section, we generalize the grouping technique as known from offline scheduling on identical machines scheduling to online scheduling on uniform machines in order to deal with general instances when rounding and preprocessing jobs as described in [Section 5.6](#). In the former technique, for a makespan or minimum completion time guess T , the jobs of processing time $< \varepsilon T$ are replaced by $\lceil (\varepsilon T)^{-1} \cdot \sum_{j: p_j < \varepsilon T} p_j \rceil$ jobs of size εT . We proceed similarly by replacing small jobs $J^{(\text{small})} := \{j \in \mathcal{J} \mid p_j < \varepsilon p_{\max}\}$ by substitute jobs of size $\varepsilon p_{\max}$. We call those substitute jobs “frames”. Let

$$F := \frac{\sum_{j: p_j \leq \varepsilon p_{\max}} p_j}{\varepsilon p_{\max}}$$

We refer to the original instance by the term *unframed instance*, whereas the one where small jobs are substituted by $f \geq F$ frames is called the *f-framed instance*. We will later see how to choose f appropriately. Naturally, $f \geq F$ should hold, but f should not become too large in comparison to F .

Before dealing with the more sophisticated problems rooted in the online setting, let us first verify that, if the set of small jobs is fixed, it does indeed suffice to work with frames and convert the resulting schedule in order to achieve the desired result. When dealing with the assignment of large jobs ($p_j \geq \varepsilon p_{\max}$), we extend the vector of job types by one additional type for the frames, and use them as the $d + 1$ -th job type in our notation.

Now, we show that if we have a schedule for the framed instance we can convert it into a schedule for the unframed instance with no larger makespan or smaller minimum completion time, while the load error increases by at most $\varepsilon p_{\max}$.

Lemma 40. *Given a schedule σ with makespan T (minimum completion time T) and for an f -framed instance, in polynomial time we can compute a schedule for the unframed instance with makespan T (minimum completion time T) and additional load error $\varepsilon p_{\max}$.*

Proof. This argument is analogous to the classical one for grouping on identical machines: A simple greedy algorithm for placing the original jobs in place of frames (e.g. list-scheduling) yields the desired result. $\square$

To be able to use the framed instance instead of the unframed one, we also need the other direction, i.e. prove that if there is a schedule with makespan T (minimum completion time T) for the unframed

instance, then there is also such a schedule with some additional load error for the framed instance. Note that the additional load error does depend on $f - F$, which we assume to be upper bounded by a constant k , and lower bounded by 0. In order to be as precise as possible, in this statement we do not state the additional load error but instead allow a specific number of frames to be accounted with processing time 0 on each machine. This number then implies that the additional load error is at most the number times $\varepsilon p_{\max}$ for approximating the makespan. For approximating the minimum completion time, the lower bound on $f - F$ is directly show the bound on the load error.

Lemma 41. *Given a schedule σ for the unframed instance with makespan T (minimum completion time T), in polynomial time we can compute a schedule σ' for the f -framed instance for any $f \in [F .. F + k]$ and $k \in \mathbb{N}_{\geq 0}$, such that σ' has makespan at most T if $1 + \left\lceil \frac{k}{m} \right\rceil$ frames per machine can be accounted with processing time 0 (σ' has minimum completion time at least $T - (1 + \left\lceil \frac{k}{m} \right\rceil) \varepsilon p_{\max}$).*

Proof. The argument here is also closely related to the classical one on identical machines. For the case of makespan minimization, we show the statement by construction a schedule σ' that uses *at least* f frames. This then shows the claim on the makespan, as the removal of frames does not increase the makespan. For the case of maximizing the minimum completion time, we show the statement by assigning exactly f frames in σ' , which is enough to show that σ' has minimum completion time at least $T - (1 + \left\lceil \frac{k}{m} \right\rceil) \varepsilon p_{\max}$. In order to construct such a schedule σ' , for the non-small jobs, we use the very same assignment as in σ . Then all that is left to do is to find an assignment of the f frames without exceeding the makespan T (subceeding the minimum completion time T). For a machine i , let $J_i^{(\text{small})}$ be the set of small jobs assigned to i under σ . We then assign

$$\phi_i := 1 + \left\lceil \frac{k}{m} \right\rceil + \left\lfloor \frac{\sum_{j \in J_i^{(\text{small})}} p_j}{\varepsilon p_{\max}} \right\rfloor$$

frames onto machine i .

We now show that completion time of i under σ' is no larger than under σ when accounting for $1 + \left\lceil \frac{k}{m} \right\rceil$ frames per machine with processing time 0, and no smaller than under σ in any case. For that, consider the difference of the load between σ' and σ :

$$\begin{aligned} 0 &\geq \sum_{j \in J_i^{(\text{small})}} p_j - \varepsilon p_{\max} \left(\left(1 + \left\lceil \frac{k}{m} \right\rceil \right) + \left\lfloor \frac{\sum_{j \in J_i^{(\text{small})}} p_j}{\varepsilon p_{\max}} \right\rfloor \right) \\ &\geq -\varepsilon p_{\max} \left(1 + \left\lceil \frac{k}{m} \right\rceil \right). \end{aligned}$$

Note that since we did not change the assignment of non-small jobs, we do not have to consider them here. Now, since we are allowed to account $1 + \left\lceil \frac{k}{m} \right\rceil$ frames per machine with processing time 0, we can conclude that σ' then has makespan at most T . We now only have to show that by our procedure, at least f frames are distributed when minimizing the makespan:

$$\begin{aligned} \sum_{i \in [m]} \phi_i &= \sum_{i \in [m]} \left(1 + \left\lceil \frac{k}{m} \right\rceil + \left\lfloor \frac{\sum_{j \in J_i^{(\text{small})}} p_j}{\varepsilon p_{\max}} \right\rfloor \right) \\ &\geq m \left\lceil \frac{k}{m} \right\rceil + \sum_{i \in [m]} \left\lfloor \frac{\sum_{j \in J_i^{(\text{small})}} p_j}{\varepsilon p_{\max}} \right\rfloor \geq k + F \geq f \quad \square \end{aligned}$$

At the same time, any machine i has at least the same load in σ' as in σ , and removing at most $1 + \left\lceil \frac{k}{m} \right\rceil$ frames per machine (until exactly f frames are placed) reduces the load by at most $(1 + \left\lceil \frac{k}{m} \right\rceil) \varepsilon p_{\max}$ per machine, leading to a minimum completion time of at least $T - (1 + \left\lceil \frac{k}{m} \right\rceil) \varepsilon p_{\max}$. This shows the statement for this case.

Now, let us focus on the online setting. More precisely, how can we ensure that we have an appropriate number F' of frames at any step, i.e. $F \leq f \leq F + \mathcal{O}(1)$, such that we have bounded amortized migration when small jobs are added/removed? For this, we have to ensure that “togglng” is not possible, e.g. if some job is added, removed, added, removed, $\dots$, this may not cause a change of f each time. We do so by maintaining the following two invariants:

Invariant 7.A. $F \leq f \leq F + 3$.

Invariant 7.B. If $\Delta f \neq 0$, then $\nabla^{(q+1)} f = \left\lceil \nabla^{(q+1)} F \right\rceil + 1$.

This directly implies the algorithmic approach: we leave f untouched until some step q , where there is a small job j inserted/removed, and afterwards [Invariant 7.A](#) would be violated if f was not changed. We then set $\nabla^{(q)} f := \left\lceil \nabla^{(q)} F \right\rceil + 1$, which directly implies that both [Invariant 7.B](#) and [Invariant 7.A](#) are always maintained.

Whenever f changes, handling this change is equivalent to adding/removing a constant number of $\varepsilon p_{\max}$ sized jobs, which is exactly our intent: in the following sections, we only have to deal with jobs whose size is at least $\varepsilon p_{\max}$ and their insertion/removal.

Lemma 42 (Amortized Change Of The Number Of Frames). *There is a step-dependent potential $\nabla^{(q)} \Phi \in \mathbb{N}_{\geq 0}$ such that $\nabla^{(0)} \Phi = 0$ and for each step where a small job J is either inserted or removed we have*

$$|\Delta \Phi| \leq 3\varepsilon p_{\max} \quad , \quad \text{and} \quad \varepsilon p_{\max} |\Delta f| + \Delta \Phi \leq 3p_j \quad .$$

Proof. For real numbers a, b , let $a \dot{-} b := \max\{a - b, 0\}$. We set

$$\nabla^{(q)}\Phi := 3\varepsilon p_{\max} \left(\left(\nabla^{(q)}f \dot{-} (\nabla^{(q)}F + 2) \right) + \left(\nabla^{(q)}F + 1 \dot{-} \nabla^{(q)}f \right) \right) .$$

The first claim holds by construction of Φ and [Invariant 7.A](#). We now prove the second claim. Observe that at most one of the summands of Φ can be non-zero (or none, but not both). Consider some fixed step $q \rightarrow q + 1$.

Case 1: $\Delta f = 0$, i.e. we did not change the number of frames. Then $f := \nabla^{(q)}f = \nabla^{(q+1)}f$ and $\nabla^{(q+1)}F = \nabla^{(q)}F \pm \frac{p_j}{\varepsilon p_{\max}}$. We then have

$$\begin{aligned} \frac{|\Delta\Phi|}{3\varepsilon p_{\max}} &= \left(f \dot{-} \left(\nabla^{(q)}F \pm \frac{p_j}{\varepsilon p_{\max}} + 2 \right) \right) + \left(\nabla^{(q)}F \pm \frac{p_j}{\varepsilon p_{\max}} + 1 \dot{-} f \right) \\ &\quad - \left(f \dot{-} (\nabla^{(q)}F + 2) \right) - \left(\nabla^{(q)}F + 1 \dot{-} f \right) \end{aligned}$$

and, since at most two of the summands in the definition of Φ can be non-zero (one positive and one negative)

$$\leq \frac{p_j}{\varepsilon p_{\max}} .$$

Multiplying both sides with $3\varepsilon p_{\max}$ then shows the claim.

Case 2: $\Delta f \neq 0$. Note that by [Invariant 7.B](#), in this case we have $\nabla^{(q+1)}f = \lceil \nabla^{(q+1)}F \rceil + 1$. This implies that

$$\begin{aligned} &\frac{\nabla^{(q+1)}\Phi}{3\varepsilon p_{\max}} \\ &= \underbrace{\left(\lceil \nabla^{(q+1)}F \rceil + 1 \dot{-} (\nabla^{(q+1)}F + 2) \right)}_{=0} + \left(\nabla^{(q+1)}F + 1 \dot{-} \nabla^{(q+1)}f \right) \\ &= \nabla^{(q+1)}F + 1 \dot{-} \left(\lceil \nabla^{(q+1)}F \rceil + 1 \right) = 0 . \end{aligned}$$

In other words: Whenever the number of frames is changed, the potential drops to 0 thereafter. This means that the potential change is exactly the value of the potential at step q .

Case 2.1: $3 \geq \Delta f > 0$, i.e. we have added some frames. Moreover, $\nabla^{(q+1)}F = \nabla^{(q)}F + \frac{p_j}{\varepsilon p_{\max}}$, as, if we needed to add a frame, this can only be due to some small job J that was just inserted.

Since we had to add frames, we know that otherwise [Invariant 7.A](#) would have been violated. In particular, this implies

$$\nabla^{(q)}f < \nabla^{(q+1)}F = \nabla^{(q)}F + \frac{p_j}{\varepsilon p_{\max}} < \nabla^{(q)}F + 1 .$$

Thus,

$$\begin{aligned} \frac{\nabla^{(q)}\Phi}{3\varepsilon p_{\max}} &= (\nabla^{(q)}F + 1) \dot{-} \nabla^{(q)}f \\ &> (\nabla^{(q)}F + 1) \dot{-} \left(\nabla^{(q)}F + \frac{p_j}{\varepsilon p_{\max}} \right) \geq 1 - \frac{p_j}{\varepsilon p_{\max}} , \end{aligned}$$

which implies $\Delta\Phi < -3\epsilon p_{\max} + 3p_j$. Moreover, due to [Invariant 7.B](#) we have

$$\nabla^{(q+1)}f = \left\lceil \nabla^{(q+1)}F \right\rceil + 1 \leq \left\lceil \nabla^{(q)}F \right\rceil + 2 \leq \nabla^{(q)}F + 3 ,$$

and thus $\Delta f \leq 3$ as $\nabla^{(q)}f \geq \nabla^{(q)}F$ by [Invariant 7.A](#). This then show the claim due to

$$\epsilon p_{\max}|\Delta f| + \Delta\Phi \leq 3\epsilon p_{\max} - 3\epsilon p_{\max} + 3p_j = 3p_j .$$

Case 2.2: $\Delta f < 0$, i.e we removed some frames. Moreover, $\nabla^{(q+1)}F = \nabla^{(q)}F - \frac{p_j}{\epsilon p_{\max}}$, as, if we needed to remove a frame, this can only be due to some small job J that was just removed.

Since we had to remove frames, we know that otherwise [Invariant 7.A](#) would have been violated. In particular, this implies

$$\nabla^{(q)}f > \nabla^{(q+1)}F + 3 = \nabla^{(q)}F - \frac{p_j}{\epsilon p_{\max}} + 3 > \nabla^{(q)}F + 2 .$$

Thus,

$$\begin{aligned} \frac{\nabla^{(q)}\Phi}{3\epsilon p_{\max}} &= \nabla^{(q)}f \div (\nabla^{(q)}F + 2) \\ &> \left(\nabla^{(q)}F - \frac{p_j}{\epsilon p_{\max}} + 3 \right) \div (\nabla^{(q)}F + 2) \geq 1 - \frac{p_j}{\epsilon p_{\max}} , \end{aligned}$$

which implies $\Delta\Phi < -3\epsilon p_{\max} + 3p_j$. Moreover, due to [Invariant 7.B](#) we have

$$\nabla^{(q+1)}f = \left\lceil \nabla^{(q+1)}F \right\rceil + 1 \geq \left\lceil \nabla^{(q)}F \right\rceil + 1 \geq \nabla^{(q)}F + 1 ,$$

and thus $\Delta f \geq -2$ as $\nabla^{(q)}f \leq \nabla^{(q)}F + 2$ by [Invariant 7.A](#).

$$\epsilon p_{\max}|\Delta f| + \Delta\Phi \leq 2\epsilon p_{\max} - 3\epsilon p_{\max} + 3p_j \leq 3p_j ,$$

as desired. $\square$

Now, we need a more complex variant of [Lemma 40](#) that ensures that a different assignment of frames and/or a different number of frames only leads to a bounded change of the assignment of the small jobs into the space occupied by frames.

Lemma 43. *Let q be a step. Assume that [Invariant 7.A](#) holds at q and $q + 1$. Let $\nabla^{(q)}\sigma$ be a schedule for the $\nabla^{(q)}f$ -framed instance and $\nabla^{(q+1)}\sigma$ for the $\nabla^{(q+1)}f$ -framed instance. Moreover, let $\nabla^{(q)}\sigma^{(\text{small})}$ be an assignment of small jobs into the space occupied by frames in $\nabla^{(q)}\sigma$, such that, for all i , we have*

$$\begin{aligned} (1 + \nabla^{(q)}\sigma_i(d+1))\epsilon p_{\max} &\geq \sum_j p_j \nabla^{(q)}\sigma_i^{(\text{small})}(j) \\ &\geq (\nabla^{(q)}\sigma_i(d+1) - 3)\epsilon p_{\max} . \end{aligned}$$

Then, in polynomial time, we can find a schedule $\nabla^{(q+1)}\sigma^{(\text{small})}$ for the small jobs at step $q + 1$ such that for all i we have

$$\begin{aligned} (1 + \nabla^{(q+1)}\sigma_i(d+1))\varepsilon p_{\max} &\geq \sum_j p_j \nabla^{(q+1)}\sigma_i^{(\text{small})}(j) \\ &\geq (\nabla^{(q+1)}\sigma_i(d+1) - 3)\varepsilon p_{\max}, \end{aligned}$$

and migration for small jobs is bounded by $2 \sum_{i \in \mathcal{M}} |\Delta\sigma_i(d+1)|$.

Proof. Let $\sigma^{(\text{small})} := \nabla^{(q)}\sigma^{(\text{small})}$. We modify $\sigma^{(\text{small})}$ in such a way that, after the modifications, setting $\nabla^{(q+1)}\sigma^{(\text{small})} := \sigma^{(\text{small})}$ returns $\nabla^{(q+1)}\sigma^{(\text{small})}$ with the required properties. Let $\mathcal{J}^-$ be the small jobs to be removed, $\mathcal{J}^+$ the small jobs to be added. Consider [Algorithm 16](#) for modifying $\sigma^{(\text{small})}$, which we execute for the given values.

Algorithm 16

```

1: procedure ONLINECONVERTFRAMEDTOUNFRAMED( $\nabla^{(q)}\sigma^{(\text{small})}$ ,
    $\mathcal{J}^-, \mathcal{J}^+$ )
2:    $\sigma^{(\text{small})} \leftarrow \nabla^{(q)}\sigma^{(\text{small})}$ 
3:   Remove all jobs in  $\mathcal{J}^-$  from  $\sigma^{(\text{small})}$ 
4:   for all  $i \in \mathcal{M}$ :  $\sigma_i(d+1)\varepsilon p_{\max} < \sum_j p_j \sigma_i^{(\text{small})}(j)$  do
5:     Remove small jobs from  $i$  until  $\sigma_i(d+1)\varepsilon p_{\max} \geq$ 
        $\sum_j p_j \sigma_i^{(\text{small})}(j)$ 
6:     Add these jobs to  $\mathcal{J}^+$ 
7:   end for
8:   while  $\mathcal{J}^+ \neq \emptyset$  do
9:     Remove a job  $J$  from  $\mathcal{J}^+$ 
10:    Place  $J$  on a machine  $i' = \max_{i \in \mathcal{M}} (\sigma_i(d+1)\varepsilon p_{\max} -$ 
        $\sum_j p_j \sigma_i^{(\text{small})}(j))$ 
11:   end while
12:    $\nabla^{(q+1)}\sigma^{(\text{small})} \leftarrow \sigma^{(\text{small})}$ 
   return  $\nabla^{(q+1)}\sigma^{(\text{small})}$ 
13: end procedure

```

After this process, $\sigma^{(\text{small})}$ contains all small jobs of the instance. Moreover, by [Invariant 7.A](#), in every step of the iterative placement of jobs in $\mathcal{J}^+$, there exists a machine i with $\sigma_i(d+1)\varepsilon p_{\max} \geq \sum_j p_j \sigma_i^{(\text{small})}(j)$. By definition, the algorithm always chooses such a machine i to place the current $J \in \mathcal{J}^+$. Thus, after placing a small job of processing time $< \varepsilon p_{\max}$ on i , $(1 + \nabla^{(q+1)}\sigma_i(d+1))\varepsilon p_{\max} \geq \sum_j p_j \nabla^{(q)}\sigma_i^{(\text{small})}(j)$ holds.

Also, if there was a machine i' with

$$(\sigma_{i'}(d+1) - 3)\varepsilon p_{\max} > \sum_j p_j \sigma_{i'}^{(\text{small})}(j)$$

after the algorithm terminates, then

$$\sigma_i(d+1)\varepsilon p_{\max} \geq \sum_j p_j \sigma_i^{(\text{small})}(j)$$

for all $i \in \mathcal{M}$, as this implies that for all i that

$$\begin{aligned} \sigma_i(d+1)\varepsilon p_{\max} - \sum_j p_j \sigma_i^{(\text{small})}(j) &\geq \sigma_{i''}(d+1)\varepsilon p_{\max} - \sum_j p_j \sigma_{i''}^{(\text{small})}(j) \\ &> \varepsilon p_{\max} \end{aligned}$$

in every iteration of the while-loop, and one job of processing time $< \varepsilon p_{\max}$ was added to any i in each iteration. This would imply that $f > F + 3$, a violation of [Invariant 7.A](#), and can therefore not occur. Thus, the process returns a schedule with

$$\begin{aligned} (1 + \nabla^{(q+1)} \sigma_i(d+1))\varepsilon p_{\max} &\geq \sum_j p_j \nabla^{(q+1)} \sigma_i^{(\text{small})}(j) \\ &\geq (\nabla^{(q+1)} \sigma_i(d+1) - 3)\varepsilon p_{\max} . \end{aligned}$$

The only jobs to cause migration are the ones added to $\mathcal{J}^+$ after previously having been removed from some machine i . Such jobs are only removed if $(1 + \sigma_i(d+1))\varepsilon p_{\max} < \sum_j p_j \sigma_i^{(\text{small})}(j)$ for machine i . This only occurs if $\Delta \sigma_i(d+1) < 0$ for i , and removing jobs of total processing time at most $|\Delta \sigma_i(d+1)| + \varepsilon p_{\max}$ is enough to get $(1 + \sigma_i(d+1))\varepsilon p_{\max} \geq \sum_j p_j \sigma_i^{(\text{small})}(j)$. Therefore, since $p_{\delta+1} = \varepsilon p_{\max}$, at most $|\Delta \sigma_i(d+1)| + \varepsilon p_{\max} \leq 2|\Delta \sigma_i(d+1)|$ jobs are migrated per machine i , which sums up to a total migration of at most $2 \sum_{i \in \mathcal{M}} |\Delta \sigma_i(d+1)|$. This shows the claim. $\square$

We are now ready to prove [Theorem 17](#), the main theorem on dynamic grouping, which is basically just a chaining of the results above. Note that [Algorithm 16](#) can be executed in time $\mathcal{O}(|I|)$.

Proof of Theorem 17. In the following, we describe our algorithmic approach. At any step q , we set f such that [Invariant 7.A](#) and [Invariant 7.B](#) are maintained. Consequently, at step $q = 0$, let $f = 1$.

We first show the statement for $\mathcal{Q}||C_{\max}$. Let σ be a schedule for $\nabla^{(q)}\nu$ for the unframed instance with makespan T . Due to [Lemma 41](#), there exists a schedule σ' for the framed instance with makespan T , if $1 + \left\lceil \frac{k}{m} \right\rceil$ frames per machine can be accounted with processing time 0. As $k \leq 3$ according to the invariants, this implies an additional load error of at most $1 + \left\lceil \frac{3}{m} \right\rceil$ per machine. We now want to go from q to $q + 1$, adding or removing some job J . If J is large, we simply use the robust algorithm to add or remove it. If J is small, we adjust f accordingly. Due to [Lemma 42](#), there is a step-dependent potential $\nabla^{(q)}\Phi \in \mathbb{N}_{\geq 0}$ such that $\nabla^{(0)}\Phi = 0$ and for each step where a small job J is either inserted or removed we have $|\Delta \Phi| \leq 3\varepsilon p_{\max}$ and $\varepsilon p_{\max}|\Delta f| + \Delta \Phi \leq 3p_j$. Thus, incorporating the frames added or removed by the change Δf using the robust algorithm has amortized migration at most $\beta(3p_j)$ and bounded reassignment potential $3\varepsilon p_{\max}$, resulting in a schedule $\bar{\sigma}'$ for the framed instance at step $q + 1$ with an additional

load error at most $1 + \left\lceil \frac{k}{m} \right\rceil$. Since, by assigning the frames at step q accordingly, all prerequisites of [Lemma 43](#) are met, we can use the Lemma to construct a schedule $\bar{\sigma}$ for the unframed instance at step $q + 1$ in polynomial time, migration bounded in $2 \sum_{i \in \mathcal{M}} |\Delta \sigma_i(d + 1)| \leq 2\beta(p_j)$. Since $(1 + \nabla^{(q+1)} \sigma_i(d + 1))\varepsilon p_{\max} \geq \sum_j p_j \nabla^{(q)} \sigma_i^{(\text{small})}(j)$, this assignment here introduces an additional load error of at most $\varepsilon p_{\max}$. Thus, we have found a schedule $\bar{\sigma}$ with amortized migration bounded in $2\beta(3p_j)$, bounded reassignment potential $3\varepsilon p_{\max}$, additional load error at most $\varepsilon p_{\max}(2 + \left\lceil \frac{3}{m} \right\rceil)$, and in time $\mathcal{O}(g \cdot \text{poly}(|I|))$.

We now show the statement for $Q||C_{\min}$. Let σ be a schedule for $\nabla^{(q)} \nu$ for the unframed instance with minimum completion time T . Due to [Lemma 41](#), there exists a schedule σ' for the framed instance with minimum completion time $T - (1 + \left\lceil \frac{k}{m} \right\rceil) \geq T - (1 + \left\lceil \frac{3}{m} \right\rceil)$, as $k \leq 3$ according to the invariants. We now want to go from q to $q + 1$, adding or removing some job J . If J is large, we simply use the robust algorithm to add or remove it. If J is small, we adjust f accordingly. Due to [Lemma 42](#), here is a step-dependent potential $\nabla^{(q)} \Phi \in \mathbb{N}_{\geq 0}$ such that $\nabla^{(0)} \Phi = 0$ and for each step where a small job J is either inserted or removed we have $|\Delta \Phi| \leq 3\varepsilon p_{\max}$ and $\varepsilon p_{\max} |\Delta f| + \Delta \Phi \leq 3p_j$. Thus, incorporating the frames added or removed by the change Δf using the robust algorithm has amortized migration at most $\beta(3p_j)$ and bounded reassignment potential $3\varepsilon p_{\max}$, resulting in a schedule $\bar{\sigma}'$ for the framed instance at step $q + 1$. Since, by assigning the frames at step q accordingly, all prerequisites of [Lemma 43](#) are met, we can use the Lemma to construct a schedule $\bar{\sigma}$ for the unframed instance at step $q + 1$ in polynomial time, migration bounded in $2 \sum_{i \in \mathcal{M}} |\Delta \sigma_i(d + 1)| \leq 2\beta(p_j)$. Since $\sum_j p_j \nabla^{(q)} \sigma_i^{(\text{small})}(j) \geq (\nabla^{(q+1)} \sigma_i(d + 1) - 3)\varepsilon p_{\max}$, this assignment here introduces an additional load error of at most $3\varepsilon p_{\max}$. Thus, we have found a schedule $\bar{\sigma}$ with amortized migration bounded in $2\beta(3p_j)$, bounded reassignment potential $3\varepsilon p_{\max}$, additional load error at most $\varepsilon p_{\max}(4 + \left\lceil \frac{3}{m} \right\rceil)$, and in time $\mathcal{O}(g \cdot |I|)$. □

5.8 ROBUST SCHEDULING WITH ADDITIVE ERROR $p_{\max}$

In this section, we finally want to introduce the robust greedy algorithm with additive error $p_{\max}$, which we use for the actual assignment of jobs on the blue machines. We first want to prove the following statement: Given a schedule σ for some job set where the completion time of every machine differs by at most $p_{\max}$, we can transform this schedule into a schedule $\bar{\sigma}$ for a different job set, where the completion time of every machine differs by at most $p_{\max}$, such that the migration is bounded only in the total processing time of difference in the two job sets, plus $p_{\max}$ times the number of differing jobs. For this, we use a simple procedure quite similar to List Scheduling [58],

which suffices to return us a schedule with the claimed properties. Let $\text{OPT}^*(C_{\max})$ the optimal objective value of an instance with $C_{\max}$ objective function, and $\text{OPT}^*(C_{\min})$ that of an instance with $C_{\min}$ objective. We then proceed to show that such schedules have makespan at most $\text{OPT}^*(C_{\max}) + p_{\max}$ and minimum completion time at least $\text{OPT}^*(C_{\min}) - p_{\max}$, bounding the error sufficiently on machines of large capacity. Let $C(\sigma, i) := p^\top \sigma_i / s_i$ is the completion time of machine i in σ .

Lemma 44 (Robust $p_{\max}$ -additive Approximation). *Let $\mathcal{M}$ be a set of machines, and let v^1, v^2 be two job sets. Let $\Delta v := v^1 - v^2$ be the difference in jobs between multiplicity vectors v^1 and v^2 . Let $\sigma: [m] \times [d] \rightarrow \mathbb{N}_{\geq 0}$ be a schedule with*

$$\max_{i, i' \in \mathcal{M}} (|C(\sigma, i) - C(\sigma, i')|) \leq p_{\max} .$$

Then, we can find a schedule $\bar{\sigma}: [m] \times [d] \rightarrow \mathbb{N}_{\geq 0}$ with

$$\max_{i, i' \in \mathcal{M}} (|C(\bar{\sigma}, i) - C(\bar{\sigma}, i')|) \leq p_{\max} ,$$

such that migration between σ and $\bar{\sigma}$ is bounded by $\sum_{j \in [d]} |\Delta v_j| (p_j + p_{\max})$.

Proof. Consider the following procedure.

Algorithm 17 Greedy procedure for scheduling jobs into machines with changed capacities

```

1: procedure ROBUSTGREEDY( $v^1, \sigma, v^2$ )
2:    $v^- = \min(v^1 - v^2, 0)$ 
3:    $v^+ = \min(v^2 - v^1, 0)$ 
4:   For all  $j \in [d]$ , remove  $v_j^-$  of type  $j$  from  $\sigma$ 
5:   while  $v_j^+ > 0$  for some  $j \in [d]$  do
6:      $v_j^+ = v_j^+ - 1$ 
7:      $\sigma(i', j) = \sigma(i', j) + 1$ , for some machine  $i' =$ 
        $\{i: \min_{i \in \mathcal{M}} (C(\sigma, i))\}$ 
8:   end while
9:   while  $i' = \max_{i \in \mathcal{M}} (C(\sigma, i)), i'' = \min_{i \in \mathcal{M}} (C(\sigma, i))$  and
        $C(\sigma, i') - C(\sigma, i'') > p_{\max}$  do
10:    For some  $j \in [d]$ 
11:       $\sigma(i', j) = \sigma(i', j) - 1$ 
12:       $\sigma(i'', j) = \sigma(i'', j) + 1$ 
13:    end while
14: end procedure

```

Let $\bar{\sigma}$ be the schedule returned by the procedure. v^+ is the set of jobs to be added, and v^- the set of jobs to be removed, respectively. Observe that $\Delta v = v^+ + v^-$. Jobs are migrated in σ in [Line 4](#), [Line 7](#),

and during each loop of [Line 9](#). [Line 4](#) and [Line 7](#) cause no migration by the definition of migration, as they only concern jobs that are added or removed. For measuring the migration caused by [Line 9](#), note the following: If $C(\sigma', i') - C(\sigma', i'') > p_{\max}$ holds for i', i'' in the modified schedule σ' before execution of the lines after [Line 9](#), this can only be due to the fact that jobs have been removed from i' in [Line 4](#) or added to i'' in [Line 7](#), as $|C(\sigma, i') - C(\sigma, i'')| \leq p_{\max}$ held before execution of [Line 9](#) by assumption. As jobs are only added to a machine i'' with minimum completion time $C(\sigma, i'')$ over all machines, and the maximum processing time of each job is at most $p_{\max}$, [Line 7](#) at most only adds jobs to i'' without violating $|C(\sigma, i') - C(\sigma, i'')| \leq p_{\max}$ if it held before.

Thus, $C(\sigma', i') - C(\sigma', i'') > p_{\max}$ can only occur due to jobs being removed from i'' by [Line 4](#). Let $v^{i''}$ be the jobs removed from i'' . It holds that $0 < v^{i''} \leq v^-$, and the total processing time of jobs removed from i'' is $\sum_{j \in [d]} v_j^{i''} p_j$. Again, as $|C(\sigma, i') - C(\sigma, i'')| \leq p_{\max}$ held before, it follows that $C(\sigma', i') - C(\sigma', i'') \leq p_{\max} + \sum_{j \in [d]} v_j^{i''} p_j$. Thus, moving a total of at most $p_{\max} + \sum_{j \in [d]} v_j^{i''} p_j$ from machine $i' = \max_{i \in \mathcal{M}}(C(\sigma', i))$ to i'' in every execution of the loop at [Line 9](#) implies $C(\sigma', i) - C(\sigma', i'') \leq p_{\max}$ for all $i \in \mathcal{M}$. As this is done for every $i'' = \min_{i \in \mathcal{M}}(C(\sigma', i))$, and there are at most $\sum_{j \in [d]} |v_j^-| \leq \sum_{j \in [d]} |\Delta v_j|$ such machines with $v^{i''} > 0$, [Line 9](#) terminates after moving jobs of total processing time at most $\sum_{j \in [d]} |\Delta v_j| (p_j + p_{\max})$, returning a schedule $\bar{\sigma}$ with $\max_{i, i' \in \mathcal{M}} (|C(\bar{\sigma}, i) - C(\bar{\sigma}, i')|) \leq p_{\max}$. This shows the claim. $\square$

Now, let OPT^{frac} be the optimal fractional schedule to an instance of $Q||C_{\max}$ and $Q||C_{\min}$ respectively, i. e. each machine of type t machines has load exactly $s_t \frac{p^T v}{s^T \mu}$ in OPT^{frac} . Note that $\text{OPT}^*(C_{\min}) \leq \text{OPT}^{\text{frac}} \leq \text{OPT}^*(C_{\max})$.

Lemma 45. *Let σ be a schedule for a multiplicity vector of jobs v and a set of machines $\mathcal{M}$, where $\max_{i, i' \in \mathcal{M}} (|C(\sigma, i) - C(\sigma, i')|) \leq p_{\max}$. Then $C_{\max}(\sigma) \leq \text{OPT}^*(C_{\max}) + p_{\max}$ and $C_{\min}(\sigma) \geq \text{OPT}^*(C_{\min}) - p_{\max}$.*

Proof. Let $i := \max_{i \in \mathcal{M}}(C(\sigma, i))$ and $i' := \min_{i \in \mathcal{M}}(C(\sigma, i))$. By definition, $C_{\max}(\sigma) = C(\sigma, i)$ and $C_{\min}(\sigma) = C(\sigma, i')$. Also, by definition of OPT^{frac} , we know that $C(\sigma, i') \leq \text{OPT}^{\text{frac}} \leq C(\sigma, i)$.

Since $C(\sigma, i) \leq C(\sigma, i') + p_{\max}$ by assumption, $C_{\max}(\sigma) = C(\sigma, i) \leq C(\sigma, i') + p_{\max} \leq \text{OPT}^{\text{frac}} + p_{\max} \leq \text{OPT}^*(C_{\max}) + p_{\max}$. Equivalently, $C(\sigma, i') \geq C(\sigma, i) - p_{\max}$ by assumption, and $C_{\min}(\sigma) = C(\sigma, i') \geq C(\sigma, i) - p_{\max} \geq \text{OPT}^{\text{frac}} - p_{\max} \geq \text{OPT}^*(C_{\min}) - p_{\max}$. This shows the claim. $\square$

Thus, using [Lemma 44](#) on machine sets where each machine has capacity at least $\ell \geq \varepsilon^{-1} p_{\max}$, the returned schedule is a $(1 + \varepsilon)$ -approximate solution for $C_{\max}$ and a $(1 - \varepsilon)$ -approximate solution for

$C_{\min}$, at every time point q with migration bounded by $p_j + p_{\max}$ for j being the type of job to be added/removed, and,

$$(1 + \varepsilon)s_t \text{OPT}^*(C_{\max}) \geq s_t \text{OPT}^*(C_{\max}) + \varepsilon\ell \geq s_t \text{OPT}^*(C_{\max}) + p_{\max} ,$$

$$(1 - \varepsilon)s_t \text{OPT}^*(C_{\min}) \leq s_t \text{OPT}^*(C_{\min}) - \varepsilon\ell \leq s_t \text{OPT}^*(C_{\min}) - p_{\max} ,$$

by to [Lemma 45](#), for every $t \in [\tau]$.

5.9 SOLVING THE ILPS

We present a way to solve both [\(ILP^{dynamic}\)](#) and [\(ILP^{dynamic}_{decr}\)](#). To be more precise: We propose algorithms that, given a state $(\alpha, \mu^{(\text{blue})}, \nu)$, either compute an optimal solution or correctly conclude that the given state is invalid. We prove a sensitivity bound for [\(ILP^{dynamic}\)](#) and [\(ILP^{dynamic}_{decr}\)](#) dependent only on the change of the state parameters $(\alpha, \mu^{(\text{blue})}, \nu)$, which is important for online scheduling with bounded migration, and argue how we can use the aforementioned algorithms, or more precisely, any algorithms solving the aforementioned problems, to find a new solution for a changed RHS with distance bounded as described in the sensitivity theorem. We then refine our approach to speed up the computation.

Recall that all machine speeds are rounded up to the next power of $(1 + \varepsilon)$. One core observation is the following: If we assign any job to a machine $i \in [m]$ with $s_i \text{OPT} < p_{\min}$, then this machine would have a completion time $> \text{OPT}$. Hence, when minimizing the makespan, we can a priori exclude all machines that are too slow by preassigning the $\mathbb{O}$ configuration to them:

Observation 1 (Support of T -valid solutions for $C_{\max}$). *If $(\alpha, \mu^{(\text{blue})}, \nu)$ is for [\(ILP^{dynamic}\)](#), then the associated solution $(x, \nu^{(\text{blue})})$ has*

$$x_{t,c} = \begin{cases} \mu_t & \text{if } c = \mathbb{O} \\ 0 & \text{otherwise} \end{cases}$$

for all $t \in [\tau]$ with $s_t T < p_{\min}$.

Similarly, when maximizing the minimum completion time, assigning any job to a machine $i \in [m]$ with $s_i \text{OPT} < p_{\min}$ results in an assignment of at least $\geq p_{\min} > s_i \text{OPT}$ total processing time for i , by definition of $p_{\min}$. We can thus make the following observation.

Observation 2 (Lower Bound on the Load for $C_{\min}$). *If $(\alpha, \mu^{(\text{blue})}, \nu)$ is valid for [\(ILP^{dynamic}\(\$C_{\min}\$ \)\)](#), then the associated solution $(x, \nu^{(\text{blue})})$ assigns jobs of total processing time at least $p_{\min}$ to machines with $s_t T < p_{\min}$.*

The solution $(x, \nu^{(\text{blue})})$ is then still the solution associated to the state $(\alpha, \mu^{(\text{blue})}, \nu)$, when all types t with $s_t T < p_{\min}$ in τ are replaced by a single (new) machine type t' with $s_{t'} T = p_{\min}$.

As a direct consequence,

- (i) by temporarily removing all red machines of types where the corresponding variables are either a priori known to be $\mathcal{O}$ (in the case of $C_{\max}$),
- (ii) or by temporarily replacing all machine types t with $s_t T < p_{\min}$ in τ by a single machine type t' with $s_{t'} T = p_{\min}$ (in the case of $C_{\min}$),

and removing all machine types that cannot be colored red without violating [Invariant 3.A](#) with either $T \leftarrow \text{OPT}^\uparrow$ (for $Q||C_{\max}$) or $T \leftarrow \text{OPT}^\downarrow$ (for $Q||C_{\min}$) before setting up the ILP formulation, the number of (red) machine types $\tau^{(\text{red})}$ is now bounded by

$$\mathcal{O}(\log_{1+\varepsilon}(\frac{\ell}{p_{\min}})) \subseteq \mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1} p_{\max}))$$

in [\(ILP^{dynamic}\)](#) and [\(ILP^{dynamic}\(\$C_{\min}\$ \)\)](#). As a consequence of rounding and scaling, $\tau^{(\text{red})}$ is bounded by $\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}))$ in [\(ILP^{dynamic}_{decr}\)](#) for either objective function. We prove the sensitivity and run time statements only for [\(ILP^{dynamic}\)](#) and [\(ILP^{dynamic}_{decr}\)](#) with objective $C_{\max}$; as both [\(ILP^{dynamic}\(\$C_{\min}\$ \)\)](#) and [\(ILP^{dynamic}_{decr}\)](#) with objective $C_{\min}$ have the same ILP parameters, the statements follow homogeneously.

Lemma 46 (Sensitivity Bounds). *Let $S = (\alpha, \mu^{(\text{blue})}, \nu)$ and $\bar{S} = (\bar{\alpha}, \bar{\mu}^{(\text{blue})}, \bar{\nu})$ be valid states with $\|\Delta \nu\|_1 + |\Delta \alpha| + \|\Delta \mu^{(\text{blue})}\|_1 \in Q_{\geq 1} \dot{\cup} \{0\}$. Then*

- (i) *The associated solutions of [\(ILP^{dynamic}\)](#) have a bounded distance from each other, i.e.*

$$\begin{aligned} \|\Delta x\|_1 + \|\Delta \nu^{(\text{blue})}\| \\ \leq 2^{\mathcal{O}(d \log(\varepsilon^{-1} p_{\max}))} \cdot \left(\|\Delta \nu\|_1 + |\Delta \alpha| + \|\Delta \mu^{(\text{blue})}\|_1 \right). \end{aligned}$$

- (ii) *The associated solutions of [\(ILP^{dynamic}_{decr}\)](#) have a bounded distance from each other, i.e.*

$$\begin{aligned} \|\Delta x\|_1 + \|\Delta \nu^{(\text{blue})}\| \\ \leq 2^{\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1}))} \cdot \left(\|\Delta \nu\|_1 + |\Delta \alpha| + \|\Delta \mu^{(\text{blue})}\|_1 \right). \end{aligned}$$

Proof. [\(ILP^{dynamic}\)](#) is an N -fold ILP, with block parameters $R = d + 1$, $H = \max(d, |\mathcal{C}(\ell)|)$, $S = 1$, $N = d + \tau$, and $\|A\|_\infty = \varepsilon^{-1} p_{\max}$. [\(ILP^{dynamic}_{decr}\)](#) is an N -fold ILP as well, with block parameters $R \in \mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}))$, $H \in 2^{\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1}))}$, $S = 1$, $N \in \mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}))$, and $\|A\|_\infty \leq \log(\varepsilon^{-1})$. The theorem by Jansen, Lassota and Rohwedder [79] ([Lemma 20](#)) states that given an optimal integer solution x to an N -fold ILP with constraint matrix A and RHS b , i.e. $Ax = b$, if changing the RHS to b' is feasible with a finite optimum,

then there is an optimal solution x' to the latter, i.e. $Ax' = b'$, such that

$$\|x - x'\|_1 \leq \|b - b'\|_1 \mathcal{O}(RS\|A\|_\infty)^{RS},$$

where R, S are block parameters and $\|A\|_\infty$ is the largest absolute value of any entry in A . In our case, for $(\text{ILP}^{\text{dynamic}})$ we have $R = d + 1$, $S = 1$, and $\|A\|_\infty \leq \varepsilon^{-1}p_{\max}$, which yields

$$\|x - x'\|_1 \leq \|b - b'\|_1 2^{\mathcal{O}(d \log(\varepsilon^{-1}p_{\max}))}.$$

and, for $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$, we have $R \in \mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}))$, $S = 1$, and $\|A\|_\infty \leq \log(\varepsilon^{-1})$, which yields

$$\|x - x'\|_1 \leq \|b - b'\|_1 2^{\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1}))}.$$

We then substitute α by $\lfloor \alpha \rfloor$ and $\bar{\alpha}$ by $\lfloor \bar{\alpha} \rfloor$, which – due to the integrality of the matrix – does neither change the (unique) optimal solution nor validity; and then apply the bound we just derived. To see that this shows the claim, note that, unless $b = b'$, which is trivial, we have

$$\begin{aligned} \|b - b'\|_1 &\leq \|\Delta v\|_1 + |\Delta \lfloor \alpha \rfloor| + \|\Delta \mu^{(\text{blue})}\|_1 \\ &\leq 1 + \|\Delta v\|_1 + |\Delta \alpha| + \|\Delta \mu^{(\text{blue})}\|_1 \\ &\leq 2 \cdot \left(\|\Delta v\|_1 + |\Delta \lfloor \alpha \rfloor| + \|\Delta \mu^{(\text{blue})}\|_1 \right) \end{aligned}$$

This proves both claims. $\square$

Also recall that for valid states, the associated solution is unique. Thus, due to this uniqueness of associated solutions, the application of the algorithm by Jansen and Rohwedder [81] to $(\text{ILP}^{\text{dynamic}})$ and $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$ directly yields an efficient algorithm to “solve” both ILPs, i.e., to either compute the associated solution or decide that the given state is invalid:

Lemma 47 (Solving The ILPs). *For a given state S and makespan T , we can decide if S is valid, and if so, compute the solution associated to S , that is, the optimal solution*

- (i) in time $2^{\mathcal{O}((d+\varepsilon^{-1} \log(\varepsilon^{-1}p_{\max})) \cdot \log(\varepsilon^{-1}p_{\max}))} + \mathcal{O}(|I|)$ for $(\text{ILP}^{\text{dynamic}})$.
- (ii) in time $2^{\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1}))} + \mathcal{O}(|I|)$ for $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$.

Proof. As argued before, we can (temporarily) reduce the number of machine types we need to consider for red machines, which amounts to $\tau^{(\text{red})} \in \mathcal{O}(\varepsilon^{-1} \cdot \log(\varepsilon^{-1}p_{\max}))$ red machines in $(\text{ILP}^{\text{dynamic}})$, and to $\tau^{(\text{red})} \in \mathcal{O}(\varepsilon^{-1} \cdot \log(\varepsilon^{-1}))$ in $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$. We then apply the algorithm by Jansen and Rohwedder [81], respectively. Its run time of $\mathcal{O}(H)^{2M} \log^2(\|b\|_\infty) + \mathcal{O}(MN)$ for an ILP of the form $Ax = b$, $x \geq 0$ with a linear objective, N variables, M constraints, and H being a

bound on the hereditary discrepancy of the constraint matrix A . In the case of $(\text{ILP}^{\text{dynamic}})$, we have

$$M \leq 1 + d + \tau^{(\text{red})} \in \mathcal{O}(d + \varepsilon^{-1} \cdot \log(\varepsilon^{-1} p_{\max})), H \in \text{poly}(\varepsilon^{-1} \cdot p_{\max}),$$

$$N \leq \tau^{(\text{red})} \ell^d \leq 2^{\mathcal{O}(d \log(\varepsilon^{-1} p_{\max}) + \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1} p_{\max}))},$$

where the second statement follows from the ℓ_1 -norm of any column in the matrix of $(\text{ILP}^{\text{dynamic}})$ being bounded in $\mathcal{O}(\ell) \subseteq (p_{\max}/\varepsilon)^{\mathcal{O}(1)}$, and a bound of H in the ℓ_1 -norm on columns by Beck and Fiala [16]. In the case of $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$, we have

$$M \leq 1 + d + \tau^{(\text{red})} \in \mathcal{O}(\varepsilon^{-1} \cdot \log(\varepsilon^{-1})), \quad H \leq \log(\varepsilon^{-1}),$$

$$N \leq \tau^{(\text{red})} |C_{\text{decr}}(u) \cup \hat{\mathcal{C}}| + \log(\varepsilon^{-1}) \leq 2^{\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1}))},$$

where the second statement follows from the ℓ_1 -norm of any column in the matrix of $(\text{ILP}^{\text{dynamic}}_{\text{decr}})$ being bounded in $\log(\varepsilon^{-1})$ after rounding, scaling and reducing, and the same bound on H in the ℓ_1 -norm of columns [16]. As $a \cdot b \leq a^2 + b^2$ for all numbers a and b , and $\log^2(\|b\|_{\infty}) \in \mathcal{O}(|I|)$, we get the claimed run times. $\square$

5.10 THE APPROXIMATION SCHEMES

We are now ready to combine all our previous results to obtain all main theorems and lemmas. We begin by proving Lemma 18.

Proof of Lemma 18. (i). For an instance I of $\text{Q}||C_{\max}$, we treat the initial job set $\mathcal{J}_0$ as adding $|\mathcal{J}_0|$ jobs to the instance in arbitrary order, beginning with an empty set. We thus start from a valid state $S = (0, \emptyset, \emptyset)$, for which we formulate $(\text{ILP}^{\text{dynamic}})$ and have a trivial $(1 + \mathcal{O}(\varepsilon))$ -competitive solution for both red and blue machines. To insert or delete jobs when going from a step q to step $q + 1$, we only use INSERT and REMOVE, respectively. This implies that at each step, we are in a valid state $S = (\alpha, \mu^{(\text{blue})}, \nu)$ for which we formulate $(\text{ILP}^{\text{dynamic}})$. Let σ be a $(1 + \mathcal{O}(\varepsilon))$ -competitive schedule at step q . To compute a solution for $q + 1$, we first use Lemma 47, (i) to compute a solution vector $(x, \nu^{(\text{blue})})$ for $(\text{ILP}^{\text{dynamic}})$ at $q + 1$. As the state S at $q + 1$ is valid, x is a $(1 + \mathcal{O}(\varepsilon))$ -competitive solution for the red machines. We then use Lemma 44 to compute a $(1 + \mathcal{O}(\varepsilon))$ -competitive solution: Let $\mathcal{M}$ be $\mu^{(\text{blue})}$ converted to a set of machines with $\mu_t^{(\text{blue})}$ machines of type t , for all $t \in [\tau]$, $\nu^1 = \nabla^{(q+1)} \nu^{(\text{blue})}$, and $\nu^2 = \nabla^{(q)} \nu^{(\text{blue})} + \sigma(\Delta \mu^{(\text{blue})})$, i.e. all blue jobs at q , as well as the jobs that are on machines that are red at q , but blue in $q + 1$, and without jobs that are on machines that are blue at q but red at $q + 1$, and current job set $\nu^{(\text{blue})}$. By definition of $\varphi^{(\text{lex})}$ and the $(1 + \mathcal{O}(\varepsilon))$ -competitiveness of σ for blue machines, $\max_{i, i' \in \mathcal{M}} |C(\sigma, i) - C(\sigma, i')| \leq p_{\max}$. Thus, the resulting schedule for blue machines is $(1 + \mathcal{O}(\varepsilon))$ -competitive by Lemma 45 and the fact that $\alpha \leq T s^{\top} \mu^{(\text{blue})}$, i.e. the total processing time of $\nu^{(\text{blue})}$ is at most

$T_S^\top \mu^{(\text{blue})}$, as well as the fact that all blue machines have capacity $\geq \ell$. This then results in a $(1 + \mathcal{O}(\varepsilon))$ -competitive solution for I .

As shown in Section 5.4, it holds that $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 + \|\Delta\nu\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max})$ for the change of the ILP parameters α , $\mu^{(\text{blue})}$ and ν from q to $q+1$ when using only INSERT and REMOVE to compute new states. Lemma 46, (i) then implies that the migration of jobs on red machines is bounded by $2^{\mathcal{O}(d \log(\varepsilon^{-1} p_{\max}))}$. The fact that on machines in $\Delta\mu^{(\text{blue})}$, there are jobs of total processing time at most $(1 + \mathcal{O}(\varepsilon))\ell$ each in σ by Invariant 4.A, implies that $\|\nu^1 - \nu^2\|_1 \leq \|\Delta\nu^{(\text{blue})}\|_1 + (1 + \mathcal{O}(\varepsilon))\ell \|\Delta\mu^{(\text{blue})}\|_1$. Using Lemma 44, the migration on blue machines is then bounded by $(\|\Delta\nu^{(\text{blue})}\|_1 + (1 + \mathcal{O}(\varepsilon))\ell \|\Delta\mu^{(\text{blue})}\|_1) \cdot (2p_{\max}) \in 2^{\mathcal{O}(d \log(\varepsilon^{-1} p_{\max}))}$ (assuming the worst-case of every migrated job having processing time 1), bounding the total migration of jobs by $2^{\mathcal{O}(d \log(\varepsilon^{-1} p_{\max}))}$. The computational work in each step is clearly dominated by solving (ILP_{dynamic}), which takes $2^{\mathcal{O}((d+\varepsilon^{-1} \log(\varepsilon^{-1} p_{\max})) \cdot \log(\varepsilon^{-1} p_{\max}))}$ time according to Lemma 47, (i).

(ii). The statement follows homogeneously when using INSERT' and REMOVE' to insert and remove jobs, respectively, as (ILP_{dynamic}($C_{\min}$)) has the same N -fold parameters as (ILP_{dynamic}), and we show in Section 5.5 that $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 + \|\Delta\nu\|_1 \leq \text{poly}(\varepsilon^{-1}, p_{\max})$ for the such computed valid states as well. Lemma 46, (i) and Lemma 47, (i) then also hold for (ILP_{dynamic}($C_{\min}$)), and computing solutions for the red machines via Lemma 47, (i) and for the blue machines via Lemma 44 then leads to $(1 - \mathcal{O}(\varepsilon))$ -competitive solutions at each step q for an instance I of $Q||C_{\min}$, with migration bounded by $2^{\mathcal{O}(d \log(\varepsilon^{-1} p_{\max}))}$ and run time in $2^{\mathcal{O}((d+\varepsilon^{-1} \log(\varepsilon^{-1} p_{\max})) \cdot \log(\varepsilon^{-1} p_{\max}))}$. $\square$

If there are no small jobs ($p_j < \varepsilon p_{\max}$) present in the instance, then we achieve an EPTAS by using the preprocessing as described in Section 5.6 to get (ILP_{dynamic}_{decr}). With this, we prove Theorem 16.

Proof of Theorem 16. After preprocessing any given instance of dynamic $Q||C_{\max}$ (dynamic $Q||C_{\min}$) as described in Section 5.6, we proceed as in the proof of Lemma 18, setting up (ILP_{dynamic}_{decr}) at each step q with the appropriate objective function for $Q||C_{\max}$ ($Q||C_{\min}$). After this preprocessing, we have $p_{\max} \in \mathcal{O}(\varepsilon^{-1})$ and the bound $|\Delta\alpha| + \|\Delta\mu^{(\text{blue})}\|_1 + \|\Delta\nu\|_1 \leq \text{poly}(\varepsilon^{-1})$, allowing us to find a $(1 + \mathcal{O}(\varepsilon))$ -competitive $((1 - \mathcal{O}(\varepsilon))$ -competitive) solution at each step q , with migration bounded by $2^{\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1}))}$ both on the red machines, solely by Lemma 46, (ii), and on the blue machines, by Lemma 46, (ii) and Lemma 44. The run time is again dominated by solving the ILP as well, which, by Lemma 47, (ii), takes $2^{\mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}) \log \log(\varepsilon^{-1}))}$ time. $\square$

For general instances, achieving an EPTAS is not so easy. We still manage to achieve a result in the direction of an EPTAS, by allowing an additional load error of $\mathcal{O}(\varepsilon p_{\max})$ on the solutions, and allowing the

migration factor to be amortized. We manage to bound the reassignment potential of the amortized migration factor in terms of $3\epsilon p_{\max}$. We do this by combining [Theorem 16](#) with our general dynamic grouping framework ([Theorem 17](#)). With this, we prove [Theorem 15](#).

Proof of [Theorem 15](#). By [Theorem 17](#), if there exists a robust algorithm for dynamic $Q||C_{\max}$ (dynamic $Q||C_{\min}$) with run time g for jobs of size $\geq \epsilon p_{\max}$, where migration is bounded by a function $\beta(p_J)$ for each job J , then there is a robust algorithm for $Q||C_{\max}$ ($Q||C_{\min}$) for all jobs with amortized migration bounded by $2\beta(3p_j)$, bounded reassignment potential $3\epsilon p_{\max}$, run time at most $\mathcal{O}(g \cdot \text{poly}(\epsilon^{-1}))$, and with additional load error at most $(4 + \lceil \frac{3}{m} \rceil)\epsilon p_{\max}$. Such an algorithm exists by [Theorem 16](#), with $g = 2^{\mathcal{O}(\epsilon^{-1}(\log \epsilon^{-1})^2)} + \mathcal{O}(|I|)$, and migration bounded by $\beta(p_j) \leq 2^{\mathcal{O}(\epsilon^{-1}(\log \epsilon^{-1})^2)}$. This proves the claim. $\square$

5.11 CONSTRUCTION OF LEGACY SCHEDULES

In this section, we want to show that a robust algorithm for the high-multiplicity encoding of an instance implies a robust algorithm for the original, non high-multiplicity instance, with the same migration factor and approximation guarantee. We call a schedule $\sigma: \mathcal{J} \rightarrow [m]$ for the original instance *legacy schedule*, and a schedule $\sigma': [m] \times [d] \rightarrow \mathbb{N}_{\geq 0}$ for the high-multiplicity instance simply *schedule*. Let $\text{type}(J)$ be a function that returns the type $j \in [d]$ of a job $J \in \mathcal{J}$. We now say a legacy schedule σ follows a schedule σ' iff, for each $i \in [m]$ and $j \in [d]$, $\sigma'(i, j) = |\{J: \sigma^{-1}(i) = J \wedge \text{type}(J) = j\}|$, i.e. each machine i has the same number of jobs of each type j in both schedules σ, σ' . Thus, if σ follows σ' , both schedules have the same objective value regarding $C_{\max}$ and $C_{\min}$. We are now ready to prove a bound on the migration for legacy schedules in the online setting as well.

Lemma 48. *Let $\mathcal{J}$ be the job set for the non high-multiplicity instance at time q , $\tilde{\mathcal{J}}$ the job set for the non high-multiplicity instance at time $\bar{q}$. For a legacy schedule $\sigma: \mathcal{J} \rightarrow [m]$ that follows some schedule σ' for q , a schedule $\bar{\sigma}'$ for $\bar{q}$ with (amortized) migration bounded by β compared to σ' , we can construct a legacy schedule $\bar{\sigma}: \tilde{\mathcal{J}} \rightarrow [m]$ that follows $\bar{\sigma}'$ in polynomial time, with (amortized) migration (compared to σ) bounded by β .*

Proof. Consider [Algorithm 18](#). First note that [Line 13](#) and [Line 20](#) can always be executed without an error: As σ is following σ' , there are exactly $\sigma'(i, j)$ jobs J of type j scheduled on i in σ . As $\bar{\sigma}'(i, j) \geq 0$ for all i, j , if $\Delta(i, j) < 0$, $|\Delta(i, j)| \leq \sigma'(i, j)$ follows. Similarly, for all $\Delta(i, j) \geq 0$, $|\Delta(i, j)|$ many jobs of type j are scheduled additionally on i in $\bar{\sigma}'$ compared to σ' . These jobs are either jobs migrated from some other machines in σ' , or jobs that are in $\tilde{\mathcal{J}}$ but not in $\mathcal{J}$. By [Line 3](#) and [Line 15](#), all such jobs are included in $\mathcal{J}^+$ when [Line 20](#) is executed, and, since σ follows σ' , there are always at least $|\Delta(i, j)|$ jobs J of type

Algorithm 18 Procedure for constructing a legacy schedule $\bar{\sigma}$ that follows a schedule $\bar{\sigma}'$, given a legacy schedule σ that follows a schedule σ' which has bounded migration β compared to $\bar{\sigma}'$

```

1: procedure LEGACYScheduleConvert( $\sigma, \sigma', \bar{\sigma}'$ )
2:    $\mathcal{J}^- \leftarrow \mathcal{J} \setminus \bar{\mathcal{J}}$ 
3:    $\mathcal{J}^+ \leftarrow \bar{\mathcal{J}} \setminus \mathcal{J}$ 
4:    $\bar{\sigma} \leftarrow \sigma$ 
5:   for all  $i \in [m], J \in \mathcal{J}^-$  do
6:     if  $\sigma(J) = i$  then
7:       Remove  $J$  from  $i$  in  $\bar{\sigma}$ 
8:     end if
9:   end for
10:   $\Delta(i, j) := \bar{\sigma}'(i, j) - \sigma'(i, j)$ 
11:  for all  $i \in [m], j \in [d]$  do
12:    if  $\Delta(i, j) < 0$  then
13:      Let  $X$  be a set of  $|\Delta(i, j)|$  jobs  $J$  of type  $j$  scheduled on  $i$ 
        in  $\sigma$ 
14:      Remove all  $J \in X$  from  $i$  in  $\bar{\sigma}$ 
15:       $\mathcal{J}^+ = \mathcal{J}^+ \cup X$ 
16:    end if
17:  end for
18:  for all  $i \in [m], j \in [d]$  do
19:    if  $\Delta(i, j) \geq 0$  then
20:      Schedule  $|\Delta(i, j)|$  jobs of type  $j$  on  $i$  in  $\bar{\sigma}$ , and remove
        these jobs from  $\mathcal{J}^+$ 
21:    end if
22:  end for return  $\bar{\sigma}$ 
23: end procedure

```

j in $\mathcal{J}^+$ at the time of execution for each pair i and j . This resulting legacy schedule $\bar{\sigma}$ then follows $\bar{\sigma}'$ by construction of the procedure. In total, the total processing time of jobs migrated from σ to $\bar{\sigma}$ is at most β , since, for every job of type j migrated from σ' to $\bar{\sigma}'$, we migrated at most one job of type j in our construction of $\bar{\sigma}$, proving the claim. The run time of this procedure is linear in $|\mathcal{J}|$ and m , and thus linear in the size $|I|$ of the non high-multiplicity instance I . $\square$

Thus, using [Lemma 48](#) to compute the legacy schedule of a schedule at each time q , we can at each time q return a legacy schedule with the same objective value regarding $C_{\max}$ and $C_{\min}$ and bound on the migration as the schedule at time q .

5.12 CONCLUSION

In this chapter, we considered the problem of scheduling jobs on uniform machines for two objectives, minimizing the makespan and

maximizing the minimum completion time, in a dynamic setting, where jobs may be inserted or deleted from the instance. For instances of these problems with $p_{\min} \geq \varepsilon p_{\max}$, we gave robust EPTASs, and, for general instances, robust EPTASs with an additional load error of $\mathcal{O}(\varepsilon p_{\max})$, as well as an amortized migration factor with bounded reassignment potential. The need for amortization is justified, as amortization is already required for dynamic $P||C_{\min}$, the special case of the problem of maximizing the minimum completion time on identical machines, as proven by Skutella and Verschae [117]. These results address open questions posed by Verschae [120] and Maack [100]. As an intermediate step, we provided EPASs for both problems, parameterized by the largest processing time $p_{\max}$ and the number of job types d . Interestingly, while introducing an additional load error, the run time of our EPTAS result is better than the run time of the currently fastest (true) EPTAS for offline $Q||C_{\max}$ [17], by a factor of $\mathcal{O}(\log^2)$ in the exponent, and matches the run time of the currently fastest EPTAS for $P||C_{\max}$ [18]. We achieved these results by introducing a partition of the machines into machines of small and large capacity (red and blue machines), capturing this idea in a novel ILP formulation with both N -fold structure and small dimension. We then showed how to adjust this partition for the changing makespans occurring in the dynamic setting in such a way that the sensitivity of the ILP, and thus the migration factor, stay bounded. We believe this idea can be used for other dynamic scheduling problems as well.

Our work suggests several open problems. One interesting possible implication of our results are potential improvements in both run time and migration factor for problems $P||C_{\max}$ and $P||C_{\min}$ in the dynamic setting. Scheduling on uniform machines is a generalization of scheduling on identical machines, and, for the case of identical machines, $\varepsilon p'_{\max} \leq \varepsilon \text{OPT}^*$ always holds, where $p'_{\max}$ is the current largest processing time in the instance (in contrast to the overall largest processing time $p_{\max}$). If one could adapt our approach to have an additional load error of $\mathcal{O}(\varepsilon p'_{\max})$ instead of $\mathcal{O}(\varepsilon p_{\max})$, this error would be negated for scheduling on identical machines, and applying such a result would be a direct improvement on the run time and migration factor of the robust EPTAS for $P||C_{\max}$ and $P||C_{\min}$ by Skutella and Verschae [117]. This requires a new partition of the jobs into large and small jobs any time $p'_{\max}$ changes by the insertion or deletion of a job however, which our current framework cannot handle directly. We thus pose this issue as an open question. Of even greater interest would be to get rid of the additional load error of $\mathcal{O}(\varepsilon p_{\max})$ altogether, to get a true robust EPTAS for general instances of $Q||C_{\max}$ and $Q||C_{\min}$, which we pose as a second open question.

BIBLIOGRAPHY

-
- [5] Fateme Abbasi, Sandip Banerjee, Jarosław Byrka, Parinya Chalermsook, Ameet Gadekar, Kamyar Khodamoradi, Dániel Marx, Roohani Sharma, and Joachim Spoerhase. Parameterized approximation schemes for clustering with general norm objectives. In *Proc. FOCS 2023*, pages 1377–1399, 2023.
 - [6] Alexander A. Ageev and Alexei E. Baburin. Approximation algorithms for UET scheduling problems with exact delays. *Oper. Res. Lett.*, 35(4):533–540, 2007.
 - [7] Alexander A. Ageev and Mikhail Ivanov. Approximating coupled-task scheduling problems with equal exact delays. In *Proc. DOOR 2016*, volume 9869 of *LNCS*, pages 259–271. Springer, 2016.
 - [8] Alexander A. Ageev and Alexander V. Kononov. Approximation algorithms for scheduling problems with exact delays. In *Proc. WAOA 2006*, volume 4368 of *LNCS*, pages 1–14. Springer, 2006.
 - [9] Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling. In *Proc. SODA 1997*, pages 493–500. ACM/SIAM, 1997.
 - [10] Matthew Andrews, Michel X Goemans, and Lisa Zhang. Improved bounds for on-line load balancing. *Algorithmica*, 23:278–301, 1999.
 - [11] Eric Angel, Evripidis Bampis, and Alexander V. Kononov. On the approximate tradeoff for bicriteria batching and parallel machine scheduling problems. *Theor. Comput. Sci.*, 306(1):319–338, 2003.
 - [12] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. STOC 1993*, page 623–631, 1993.
 - [13] Yossi Azar and Leah Epstein. Approximation schemes for covering and scheduling in related machines. In *Proc. APPROX 1998*, pages 39–47. Springer, 1998.
 - [14] Evripidis Bampis and Alexander V. Kononov. Bicriteria approximation algorithms for scheduling problems with communications delays. *J. Sched.*, 8(4):281–294, 2005.

- [15] Amotz Bar-Noy, Ari Freund, and Joseph Naor. New algorithms for related machines with temporary jobs. *J. Sched.*, 3:259–272, 2000.
- [16] József Beck and Tibor Fiala. “integer-making” theorems. *Discret. Appl. Math.*, 3(1):1–8, 1981.
- [17] Sebastian Berndt, Hauke Brinkop, Klaus Jansen, Matthias Mnich, and Tobias Stamm. New support size bounds for integer programming, applied to makespan minimization on uniformly related machines. In *Proc. ISAAC 2023*, volume 283 of *LIPICs*, pages 13:1–13:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [18] 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, 2022.
- [19] Sebastian Berndt, Valentin Dreismann, Kilian Grage, Klaus Jansen, and Ingmar Knof. Robust online algorithms for certain dynamic packing problems. In *Proc. WAOA 2019*, volume 11926 of *LNCS*, pages 43–59. Springer, 2019.
- [20] Sebastian Berndt, Franziska Eberle, and Nicole Megow. Online load balancing with general reassignment cost. *Oper. Res. Lett.*, 50(3):322–328, 2022.
- [21] Sebastian Berndt, Kilian Grage, Klaus Jansen, Lukas Johannsen, and Maria Kosche. Robust online algorithms for dynamic choosing problems. In *Proc. CiE 2021*, volume 12813 of *LNCS*, pages 38–49. Springer, 2021.
- [22] Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. Fully dynamic bin packing revisited. *Math. Program.*, 179(1–2):109–155, jan 2020.
- [23] S. Bessy and R. Giroudeau. Parameterized complexity of a coupled-task scheduling problem. *J. Sched.*, 22(3):305–313, 2019.
- [24] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Kráľovič, Richard Kráľovič, and Tobias Mömke. On the advice complexity of online problems. In *Proc. ISAAC 2009*, page 331–340. Springer, 2009.
- [25] Moritz Buchem, Lars Rohwedder, Tjark Vredeveld, and Andreas Wiese. Additive Approximation Schemes for Load Balancing Problems. In *Proc. ICALP 2021*, volume 198 of *LIPICs*, pages 42:1–42:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

- [26] Mihai Bundeineddoiu, Sarel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proc. STOC 2002*, page 250–257. ACM, 2002.
- [27] József Békési, György Dósa, and Gábor Galambos. A first fit type algorithm for the coupled task scheduling problem with unit execution time and two exact delays. *Eur. J. Oper. Res.*, 297(3):844–852, 2022.
- [28] Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proc. STOC 2001*, page 84–93. ACM, 2001.
- [29] Bo Chen and Xiandong Zhang. Scheduling coupled tasks with exact delays for minimum total job completion time. *J. Sched.*, 24(2):209–221, 2021.
- [30] Hua Chen, Lin Chen, and Guochuan Zhang. Fpt algorithms for a special block-structured integer program with applications in scheduling. *Math. Prog.*, 2024.
- [31] Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of approximation schemes for the classical scheduling problem. In Chandra Chekuri, editor, *Proc. SODA 2014*, pages 657–668. SIAM, 2014.
- [32] John J. Clifford and Marc E. Posner. Parallel machine scheduling with high multiplicity. *Math. Program.*, 89(3):359–383, 2001.
- [33] A. Condotta and N. V. Shakhlevich. Scheduling coupled-operation jobs with exact time-lags. *Discrete Appl. Math.*, 160(16–17):2370–2388, 2012.
- [34] Michele Conforti, Gerard Cornuejols, and Giacomo Zambelli. *Integer Programming*. Springer, 2014.
- [35] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. STOC 1971*, page 151–158. ACM, 1971.
- [36] William J. Cook, Albertus M. H. Gerards, Alexander Schrijver, and Éva Tardos. Sensitivity theorems in integer linear programming. *Math. Program.*, 34(3):251–264, 1986.
- [37] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.
- [38] 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 *Proc. SODA 2021*, pages 1666–1681. SIAM, 2021.

- [39] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Loksh-tanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 1st edition, 2015.
- [40] Kalyanmoy Deb. Multi-objective optimization. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 403–449, 2014.
- [41] 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.
- [42] Jérôme Dohrau. Online makespan scheduling with sublinear advice. In *Proc. SOFSEM 2015*, volume 8939 of *LNCS*, pages 177–188. Springer, 2015.
- [43] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness i: Basic results. *SICOMP*, 24(4):873–921, 1995.
- [44] Tomás Ebenlendr and Jirí Sgall. A lower bound on deterministic online algorithms for scheduling on related machines without preemption. *TOCS*, 56(1):73–81, 2015.
- [45] Thorsten Ehlers and Klaus Jansen. Online-scheduling on identical machines with bounded migration. In *Proc. SYNASC 2013*, pages 361–366. IEEE Computer Society, 2013.
- [46] Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster Algorithms for Integer Programs with Block Structure. In *Proc. ICALP 2018*, volume 107 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.
- [47] Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006.
- [48] Moustafa Elshafei, Hanif D. Sherali, and J. Cole Smith. Radar pulse interleaving for multi-target tracking. *Nav. Res. Logist. Q.*, 51:72–94, 2004.
- [49] Leah Epstein and Asaf Levin. A robust aptas for the classical bin packing problem. *Math. Program.*, 119:33–49, 06 2009.
- [50] A. Farina and P. Neri. Multitarget interleaved tracking for phased-array radar. *IEE PROC-F*, 127:312–318(6), 1980.
- [51] Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-dynamic bin packing with little repacking. In *Proc. ICALP*

- 2018, volume 107 of *LIPICs*, pages 51:1–51:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- [52] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Can. J. Math.*, 8:399–404, 1956.
 - [53] Ayoub Foussoul, Vineet Goyal, and Amit Kumar. Fully-dynamic load balancing. In *Proc. IPCO 2024*, page 182–195. Springer, 2024.
 - [54] Waldo Gálvez, José A. Soto, and José Verschae. Symmetry exploitation for online machine covering with bounded migration. *TALG*, 16(4), July 2020.
 - [55] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.
 - [56] Michel X. Goemans and Thomas Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proc. SODA 2014*, pages 830–839. SIAM, 2014.
 - [57] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, 5:287–326, 1979.
 - [58] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAP*, 17(2):416–429, 1969.
 - [59] Frieda Granot and Jadranka Skorin-Kapov. On polynomial solvability of the high multiplicity total weighted tardiness problem. *Discrete Appl. Math.*, 41(2):139–146, 1993.
 - [60] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proc. STOC 2017*, page 537–550. ACM, 2017.
 - [61] Sushmita Gupta, Pallavi Jain, Sanjay Seetharaman, and Meirav Zehavi. A simple algorithm for combinatorial n-fold ilps using the steinitz lemma. *CoRR*, abs/2507.03766, 2025. To appear in *Proc. IPEC 2025*.
 - [62] Hatem Hadda and Mhamed Abdessalem. New characterizations for the identical coupled tasks scheduling problem. *RAIRO Oper. Res.*, 57:325–332, 2023.
 - [63] Toshio Hamada. A bayesian sequential single machine batching and scheduling problem with random setup time. *J. Oper. Res. Soc. Jpn.*, 53(1):79–100, 2010.

- [64] Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. n -fold integer programming in cubic time. *Math. Program.*, 137(1-2):325–341, 2013.
- [65] Dorit S. Hochbaum. Approximation algorithms for np-hard problems. *SIGACT News*, 28(2):40–52, 1997.
- [66] Dorit S. Hochbaum and Ron Shamir. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Oper. Res.*, 39(4):648–653, 1991.
- [67] Dorit S. Hochbaum, Ron Shamir, and J. George Shanthikumar. A polynomial algorithm for an integer quadratic non-separable transportation problem. *Math. Prog.*, 55:359–371, 1992.
- [68] Dorit S. Hochbaum and J. George Shanthikumar. Convex separable optimization is not much harder than linear optimization. *J. ACM.*, 37(4):843–862, 1990.
- [69] 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.
- [70] Han Hoogeveen. Multicriteria scheduling. *Eur. J. Oper. Res.*, 167:592–623, 12 2005.
- [71] Yumei Huo, Haibing Li, and Hairong Zhao. Minimizing total completion time in two-machine flow shops with exact delays. *Comput. Oper. Res.*, 36(6):2018–2030, 2009.
- [72] F. J. Hwang and B. M. T. Lin. Coupled-task scheduling on a single machine subject to a fixed-job-sequence. *Comput. Ind. Eng.*, 60(4):690–698, 2011.
- [73] R. Impagliazzo and R. Paturi. Complexity of k -sat. In *Proc. CCC 1999*, pages 237–240, 1999.
- [74] Zoran Ivković and Errol L. Lloyd. Fully dynamic bin packing. In *Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz*, pages 407–434. Springer, 2009.
- [75] Klaus Jansen, Kai Kahler, Lis Piroton, and Malte Tutas. Improving the parameter dependency for high-multiplicity scheduling on uniform machines. *CoRR*, abs/2409.04212, 2025. To appear in Proc. IPEC 2025.
- [76] Klaus Jansen, Kai Kahler, and Esther Zwanger. Exact and approximate high-multiplicity scheduling on identical machines. In *Proc. CIAC 2025*, volume 15679 of *LNCS*, pages 1–17. Springer, 2025.

- [77] 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.
- [78] Klaus Jansen and Felix Land. Non-preemptive scheduling with setup times: A PTAS. In *Proc. EUROPAR 2016*, volume 9833 of *LNCS*, pages 159–170. Springer, 2016.
- [79] Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. Near-linear time algorithm for n -fold ilps via color coding. *SIDMA*, 34(4):2282–2299, 2020.
- [80] Klaus Jansen and Marten Maack. An EPTAS for scheduling on unrelated machines of few different types. *Algorithmica*, 81(10):4134–4164, 2019.
- [81] Klaus Jansen and Lars Rohwedder. On integer programming, discrepancy, and convolution. *Math. Oper. Res.*, 2022.
- [82] Xiaojuan Jiang, Kangbok Lee, and Michael L Pinedo. Approximation algorithms for bicriteria scheduling problems on identical parallel machines for makespan and total completion time. *Eur. J. Oper. Res.*, 305(2):594–607, 2023.
- [83] Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.
- [84] Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. *J. Comput. Syst. Sci.*, 81(8):1665–1677, 2015.
- [85] Mostafa Khatami, Daniel Oron, and Amir Salehipour. Scheduling coupled tasks on parallel identical machines. *Optim. Lett.*, 18(4):991–1003, 2024.
- [86] Mostafa Khatami and Amir Salehipour. A binary search algorithm for the general coupled task scheduling problem. *4OR*, 19:1–19, 2021.
- [87] Mostafa Khatami and Amir Salehipour. Coupled task scheduling with time-dependent processing times. *J. Sched.*, 24:223–236, 2021.
- [88] Mostafa Khatami, Amir Salehipour, and T.C.E. Cheng. Coupled task scheduling with exact delays: Literature review and models. *Eur. J. Oper. Res.*, 282(1):19–39, 2020.
- [89] Dusan Knop and Martin Koutecký. Scheduling kernels via configuration LP. In *Proc. ESA 2022*, volume 244 of *LIPIcs*, pages 73:1–73:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

- [90] Dušan Knop and Martin Koutecký. Scheduling meets n -fold integer programming. *J. Sched.*, 21(5):493–503, October 2018.
- [91] Dušan Knop, Michał Pilipczuk, and Marcin Wrochna. Tight complexity lower bounds for integer linear programming with few constraints. *ACM Trans. Comput. Theory*, 12(3), June 2020.
- [92] Dušan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Parameterized complexity of configuration integer programs. *Oper. Res. Lett.*, 49(6):908–913, 2021.
- [93] Fanyu Kong, Cuixia Miao, Yujia Huo, and Jiaxin Song. Single-machine scheduling with coupled task and rejection. *Math. Found. Comput.*, 7(3):328–335, 2024.
- [94] Martin Koutecký and Johannes Zink. Complexity of scheduling few types of jobs on related and unrelated machines. In *Proc. ISAAC 2020*, volume 181 of *LIPICs*, pages 18:1–18:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- [95] Wiesław Kubiak. A note on scheduling coupled tasks for minimum total completion time. *Ann. Oper. Res.*, pages 1–4, 04 2022.
- [96] Jan Karel Lenstra and David B. Shmoys. Elements of scheduling. *CoRR*, abs/2001.06005, 2020.
- [97] Joseph Y.-T. Leung, Haibing Li, and Hairong Zhao. Scheduling two-machine flow shops with exact delays. *Int. J. Found. Comput. Sci.*, 18:341–359, 2007.
- [98] Richard J. Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proc. EC 2004*, pages 125–131. ACM, 2004.
- [99] Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min k -cut. *SICOMP*, 53(6), 2020.
- [100] Marten Maack. Online load balancing on uniform machines with limited migration. *Oper. Res. Lett.*, 51(3):220–225, may 2023.
- [101] Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. Non-preemptive scheduling on machines with setup times. In *Proc. WADS 2015*, volume 9214 of *LNCS*, pages 542–553. Springer, 2015.
- [102] Dániel Marx. Parameterized Complexity and Approximation Algorithms. *Comput. J.*, 51(1):60–78, 07 2007.

- [103] S. Thomas McCormick, Scott R. Smallwood, and Frits C. R. Spieksma. A polynomial algorithm for multiprocessor scheduling with two job lengths. *Math. Oper. Res.*, 26(1):31–49, 2001.
- [104] Robert McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6(1):1–12, October 1959.
- [105] Clyde L. Monma and Chris N. Potts. On the complexity of scheduling with batch setup times. *Oper. Res.*, 37(5):798–804, 1989.
- [106] 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.
- [107] A. J. Orman and C. N. Potts. On the complexity of coupled-task scheduling. *Discrete Appl. Math.*, 72(1-2):141–154, 1997.
- [108] Marc E. Posner. The complexity of earliness and tardiness scheduling problems under id-encoding. 1985.
- [109] Harilaos N. Psaraftis. A dynamic programming approach for sequencing groups of identical jobs. *Oper. Res.*, 28(6):1347–1359, 1980.
- [110] Marc P. Renault, Adi Rosén, and Rob van Stee. Online algorithms with advice for bin packing and scheduling problems. *Theor. Comput. Sci.*, 600:155–170, 2015.
- [111] Lars Rohwedder. Eth-tight FPT algorithm for makespan minimization on uniform machines. In *Proc. ICALP 2025*, volume 334 of *LIPICs*, pages 126:1–126:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.
- [112] Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, may 2009.
- [113] Petra Schuurman and Gerhard J. Woeginger. Preemptive scheduling with job-dependent setup times. In *Proc. SODA 1999*, pages 759–767. ACM/SIAM, 1999.
- [114] Jeffrey Shallit. Origins of the analysis of the euclidean algorithm. *Historia Mathematica*, 21(4):401–419, 1994.
- [115] Roy D. Shapiro. Scheduling coupled tasks. *Nav. Res. Logist. Q.*, 27(3):489–498, 1980.
- [116] G. Simonin, R. Giroudeau, and J.C. König. Complexity and approximation for scheduling problem for a torpedo. *Comput. Ind. Eng.*, 61(2):352–356, 2011.

- [117] Martin Skutella and José Verschae. Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. *Math. Oper. Res.*, 41(3):991–1021, 2016.
- [118] Cliff Stein and Joel Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Oper. Res. Lett.*, 21(3):115–122, 1997.
- [119] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.
- [120] José Verschae. *The Power of Recourse in Online Optimization: Robust Solutions for Scheduling, Matroid and MST Problems*. PhD thesis, Technical University of Berlin, 2012.
- [121] Jeffery Westbrook. Load balancing for response time. *J. Algorithms*, 35(1):1–16, 2000.
- [122] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 1st edition, 2011.
- [123] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.
- [124] Wenxun Xing and Jiawei Zhang. Parallel machine scheduling with splitting jobs. *Discret. Appl. Math.*, 103(1-3):259–269, 2000.
- [125] Wenci Yu, Han Hoogeveen, and Jan Karel Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *J. Sched.*, 7(5):333–348, 2004.

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.

Diese Arbeit ist weder ganz noch in Teilen an anderer Stelle im Rahmen eines Prüfungsverfahrens vorgelegt worden. Teile wurden, wie in der Arbeit gekennzeichnet, im Rahmen wissenschaftlicher Veröffentlichungen publiziert.

Hamburg, Dezember 2025

David Simon Fischer

ERKLÄRUNG ÜBER DEN EIGENANTEIL AN PUBLIKATIONEN MIT MEHREREN AUTOREN

Bei der Arbeit “Robust scheduling on uniform machines” [1] (Kapitel 5) hat Herr Brinkop einen ersten Entwurf bezüglich des Abschnittes “A (Dynamic) Grouping Technique” eigenständig erstellt. Die übrigen Ideen, Beweise und den Aufschrieb dieser Arbeit haben Herr Fischer und Herr Brinkop gleichmäßig ausgearbeitet.

Die Ideen, Beweise und Aufschrieb der Arbeit “Approximation algorithms for coupled task scheduling minimizing the sum of completion times” [2] (Kapitel 2) hat Herr Fischer zu mehr als der Hälfte ausgearbeitet.

Die Ideen und Beweise der Abschnitte 3 - 5 der Arbeit “High multiplicity scheduling on uniform machines in fpt-time” [3] und den ersten Aufschrieb hat hauptsächlich Herr Brinkop ausgearbeitet. Herr Fischer und Herr Brinkop haben gemeinsam die Ideen, Beweise und den Aufschrieb der Arbeit vereinfacht und verbessert, sowie die Ideen, Beweise und den Aufschrieb von Abschnitt 6, insbesondere Abschnitt 6.2, gemeinsam ausgearbeitet. Die hier verwendeten Ergebnisse der genannten Veröffentlichung beinhalten hauptsächlich Abschnitt 6.2, und finden sich in Kapitel 4 der Thesis. Kapitel 4 enthält zudem bisher unveröffentlichte Ideen und Beweise, die in gleichmäßiger Ausarbeitung von Herrn Fischer, Herrn Schumacher, Herrn Ohnesorge, und Frau Pirotton entstanden sind. Den Aufschrieb dieser hat Herr Fischer fast vollständig alleine erstellt.

Die Ideen und Beweise der Arbeit “A $(4/3 + \varepsilon)$ -approximation for preemptive scheduling with batch setup times” [4] (Kapitel 3) hat Herr Fischer zu mehr als der Hälfte ausgearbeitet. Den Aufschrieb dieser Arbeit hat Herr Fischer zum Großteil eigenständig erstellt.

Die Arbeiten [1], [3] und [4] sind jeweils unter Betreuung und mit Unterstützung von Klaus Jansen, des Betreuers von Herrn Fischer, entstanden. Die Arbeit [2] ist unter Betreuung und mit Unterstützung von Matthias Mnich entstanden.

COLOPHON

This document was typeset using the typographical look-and-feel `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 L^AT_EX and L^yX:

<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 December 17, 2025 (`classicthesis` v4.6).