

Exact and Approximate High-Multiplicity Scheduling on Identical Machines

Klaus Jansen ✉

Kiel University, Department of Computer Science, Germany

Kai Kahler ✉

Kiel University, Department of Computer Science, Germany

Esther Zwanger ✉

Kiel University, Department of Computer Science, Germany

Abstract

Goemans and Rothvoss (SODA'14) gave a framework for solving problems which can be described as finding a point in $\text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$, where $P, Q \subset \mathbb{R}^N$ are (bounded) polyhedra. The running time for solving such a problem is $\langle P \rangle^{2^{O(N)}} \langle Q \rangle^{O(1)}$. This framework can be used to solve various scheduling problems, but the encoding length $\langle P \rangle$ usually involves parameters like the makespan or deadlines (which can be very large compared to the processing times). We describe three tools to improve the framework by Goemans and Rothvoss:

- Problem-specific preprocessing can be used to greatly reduce $\langle P \rangle$.
- By solving a certain LP relaxation, one can obtain bounds for the points in P . Combined with the classical result by Frank and Tardos (J. Comb. '87), these yield a more compact encoding of P in general.
- A result by Jansen and Klein (SODA'17) changes the running time of the algorithm by Goemans and Rothvoss to $|V|^{2^{O(N)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)}$, where V is the set of vertices of the convex hull of $P \cap \mathbb{Z}^N$. We provide a new bound for $|V|$ that is similar to the one by Berndt et al. (SOSA'21) but better for our setting; this gives an alternative way to improve the framework.

In particular, applied to the scheduling problems $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$, these tools improve the running time from $(\log(C_{\max}))^{2^{O(d)}} \langle I \rangle^{O(1)}$ to the possibly much better $(\log(p_{\max}))^{2^{O(d)}} \langle I \rangle^{O(1)}$. Here, p_{\max} is the largest processing time, d is the number of different processing times, C_{\max} is the makespan and $\langle I \rangle$ is the encoding length of the instance. This running time can be shown to be fixed-parameter tractable (FPT) w.r.t. parameter d if p_{\max} is given in unary. We show that similar results can also be obtained for other scheduling problems, MINSUM-WEIGHTED-BINPACKING and a special case of n -fold ILPs. Moreover, we show how the balancing result by Govzmann et al. can be used to speed up an additive approximation scheme by Buchem et al. (ICALP'21) in the high-multiplicity setting.

On the complexity side, we use reductions from the literature to provide new parameterized lower bounds for $P||C_{\max}$ and to show that the improved running time of the additive approximation algorithm is probably optimal. Finally, we show that the big open question asked by Mnich and van Bevern (Comput. Oper. Res. '18) whether $P||C_{\max}$ is FPT w.r.t. the number of job types d has the same answer as the question whether $Q||C_{\max}$ is FPT w.r.t. the number of job and machine types $d + \tau$ (all in high-multiplicity encoding). The same holds for objective C_{\min} .

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases scheduling, parameterized complexity, approximation algorithms, high-multiplicity, bin packing

Funding Supported by the German Research Foundation (DFG) project JA 612/25-1.

1 Introduction

Distributing jobs or tasks among workers or machines is one of the most natural optimization problems arising in practice. Such *scheduling problems* have been systematically studied for

over half a century (see e.g. [33] for a great introduction to scheduling). Different machine models, job characteristics and objectives make for a plethora of different problems; some of them can be solved efficiently by scheduling jobs in a certain order, but many are NP-hard. Hence, a large part of research on scheduling is concerned with approximation algorithms; on the other hand, there are also many pseudo-polynomial algorithms. Another – regarding the whole history of scheduling – more recent research direction are *parameterized algorithms*. Such algorithms encapsulate the complexity in certain parameters of the problem, and should these parameters be quite small, the algorithms can be very efficient [11].

In this work, we primarily focus on the parameter d , the number of different *job types*. The exact definition may vary as the problems become more complex (with weights, classes or deadlines), but in a simple setting where a job j only has a processing time p_j , the parameter d describes the number of different processing times. This parameter choice seems natural in an industrialized world, where workers and machines have a large degree of specialization and hence do not work on too many different types of jobs or tasks. We use the standard three-field notation introduced by Graham et al. [20]; for a detailed description of our problems and parameters, see Section 2.

With the result by Goemans and Rothvoss [18] (or the one by Jansen and Klein [23]), one can solve problems that can be modelled by two polytopes P and Q such that solutions of the problem directly correspond to a point $y \in \text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$. Many scheduling problems fall into this category, e.g. makespan minimization on identical machines ($P||C_{\max}$). However, when described by such a PQ-representation, the makespan C_{\max} appears in the system $Ax \leq b$ that represents the polytope P and thus greatly influences the running time.

Our Contributions

We describe three tools that can be used to improve the framework by Goemans and Rothvoss [18] such that the running time does not depend on $\|b\|_{\infty}$, which in the case of scheduling problems eliminates parameters like the makespan or the largest due date from the running time.

The first tool (covered in Section 3) is a balancing result by Govzmann et al. [19] that allows pre-scheduling many of the jobs, leaving us with a makespan value that is bounded by $2dp_{\max}$, where p_{\max} is the largest processing time. Applying this preprocessing before using the algorithm by Goemans and Rothvoss yields the following running times for solving $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$:

► **Theorem 1.** *The optimization problem $P||\{C_{\max}, C_{\min}\}$ can be solved in time $(d \log(p_{\max}) + \log(2dp_{\max}^2))^{2^{O(d)}} \langle I \rangle^{O(1)}$ and the optimization problem $P||C_{\text{envy}}$ can be solved in time $p_{\max}(d \log(p_{\max}) + \log(2dp_{\max}^2))^{2^{O(d)}} \langle I \rangle^{O(1)}$.*

We also show lower bounds for $P||C_{\max}$ that somewhat resemble the running time of our algorithm, though there is still a gap (for a proof, see Appendix D):

► **Theorem 2.** *Let $\varepsilon > 0$. Unless the ETH fails, $P||C_{\max}$ cannot be solved in time $\langle I \rangle^{O(d^{1-\varepsilon})} p_{\max}^{O(1)}$, $\langle I \rangle^{O(1)} d^{O(d^{1-\varepsilon})} p_{\max}^{O(1)}$ or $\langle I \rangle^{o(\frac{d}{\log(d)})} p_{\max}^{O(1)}$.*

Note that the encoding length $\langle I \rangle$ includes $\log(p_{\max})$ and d . In Appendix B, we show that the balancing result can be used to speed up an additive approximation scheme by Buchem et al. [4] in the case where the number of jobs $\|n\|_1$ is larger than the number of machines m times p_{\max} :

► **Theorem 3.** *There is an additive approximation scheme for $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$ with error at most εp_{\max} that has running time $(mp_{\max})^{O(\frac{1}{\varepsilon})}$.*

In Appendix C, we give a matching lower bound for objective C_{\max} via ETH:

► **Theorem 4.** *Let $\delta > 0$. Then $P||C_{\max}$ cannot be approximated with additive error at most εp_{\max} in time $(mp_{\max})^{O((\frac{1}{\varepsilon})^{1-\delta})}$, unless the ETH fails.*

The second tool (covered in Section 4) involves solving a special relaxation of an integer linear program (ILP) associated with a given PQ-representation. A proximity result by Cslovjcek et al. [9] then gives an upper bound for the points in P , which allows us to reduce the coefficients in the system $Ax \leq b$ describing P with the famous result by Frank and Tardos [17]. Together, this then yields an algorithm for solving PQ-representations in a running time independent of $\|b\|_{\infty}$:¹

► **Theorem 5.** *If a problem has a PQ-representation (P, Q, m) with $P = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(P)}x = b^{(P)}\}$ and $Q = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(Q)}x = b^{(Q)}\}$ with $A^{(P)} \in \mathbb{Z}^{M^{(P)} \times N}$, $A^{(Q)} \in \mathbb{Z}^{M^{(Q)} \times N}$, $b^{(P)} \in \mathbb{Z}^{M^{(P)}}$ and $b^{(Q)} \in \mathbb{Z}^{M^{(Q)}}$, it can be solved in time*

$$((M^{(P)}M^{(Q)} \log(\max\{\|A^{(P)}\|_{\infty}, \|A^{(Q)}\|_{\infty}\}))^{2^{O(N)}} + 2^{O((M^{(Q)})^2)})(\langle P \rangle \langle Q \rangle \log(m))^{O(1)}.$$

The third tool (covered in Section 5) is an upper bound for the number of vertices of the integer hull of a polytope that is similar to the one by Berndt et al. [2] but better for our specific purpose:

► **Theorem 6.** *The number of vertices of the integer hull of a polytope $P = \{x \in \mathbb{R}_{\geq 0}^N \mid Ax = b\}$ is bounded by $N^M O(M \log(M\Delta))^N$, where $A \in \mathbb{Z}^{M \times N}$ and $\Delta = \|A\|_{\infty}$.*

Note that this does not depend on $\|b\|_{\infty}$. Previously known bounds either depend on $\|b\|_{\infty}$ or are exponential in $\log(\|A\|_{\infty})$. Combined with the algorithm by Jansen and Klein [23], this gives an alternative way of solving PQ-representations:

► **Theorem 7.** *If a problem has a PQ-representation (P, Q, m) with $P = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(P)}x = b^{(P)}\}$ and $Q = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(Q)}x = b^{(Q)}\}$ with $A^{(P)} \in \mathbb{Z}^{M^{(P)} \times N}$, $A^{(Q)} \in \mathbb{Z}^{M^{(Q)} \times N}$, $b^{(P)} \in \mathbb{Z}^{M^{(P)}}$ and $b^{(Q)} \in \mathbb{Z}^{M^{(Q)}}$, it can be solved in time*

$$(N^{M^{(P)}} (M^{(P)} \log(\|A^{(P)}\|_{\infty}))^{2^{O(N)}} (\langle P \rangle \langle Q \rangle \log(m))^{O(1)}.$$

Both Theorem 5 and Theorem 7 can be used to obtain algorithms for various scheduling problems that are more efficient than a straightforward application of the results by Goemans and Rothvoss [18] and Jansen and Klein [23] with previously known bounds for the vertices of the integer hull. In Appendix A, we give PQ-representations for various scheduling problems, MINSUM-WEIGHTED-BINPACKING (MSWBP) and uniform n -fold ILPs.

The theorems also yield $(\log(p_{\max}))^{2^{O(d)}} \langle I \rangle^{O(1)}$ -time (resp. $p_{\max} (\log(p_{\max}))^{2^{O(d)}} \langle I \rangle^{O(1)}$) algorithms for $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$, but with worse constants than in Theorem 1. One might ask why such running times are interesting, as the parameters p_{\max} and d are still entangled. But with an inequality that can be found in an exercise from [11] (Hint 3.18) and has been used by Koutecký and Zink [31], one can bound $(\log(\Delta))^{2^{O(d)}}$ by $2^{2^{O(d)}} \Delta^{O(1)}$ (see Lemma 48). So Theorem 1 almost answers the open question by Mnich and van Bevern [35] for an algorithm solving $P||C_{\max}$ that is fixed-parameter tractable (FPT) w.r.t. d . This question had been partially answered by Koutecký and Zink [31], who gave an $f(d)n^{o(1)} \langle I \rangle^{O(1)}$ -time

¹ Of course, the entries in b still appear in the encoding length of P , but only logarithmically.

algorithm, where f is some computable function. Actually, one can analyze the function $(\log(\Delta))^{2^{O(d)}}$ even further. Define the tower function for any $\alpha \in \mathbb{N}$, $\beta \in \mathbb{R}$ recursively as follows: $\text{Tower}(\alpha, \beta) = \beta$ if $\alpha = 0$ and $\text{Tower}(\alpha, \beta) = 2^{\text{Tower}(\alpha-1, \beta)}$ otherwise. For example, $\text{Tower}(1, \beta) = 2^\beta$ and $\text{Tower}(2, \beta) = 2^{2^\beta}$. We generalize the inequality used by Koutecký and Zink [31] to the following:

► **Lemma 8.** *For every $\alpha, \beta, \gamma \in \mathbb{N}_{>0}$, we have $\beta^\gamma \leq \max\{\text{Tower}(\alpha, \gamma)^\gamma, \beta^{\log^{(\alpha)}(\beta)}\}$.*

Proof. Case 1: $\gamma \geq \log^{(\alpha)}(\beta)$. Then $\text{Tower}(\alpha, \gamma) \geq \beta$ and we get $\beta^\gamma \leq \text{Tower}(\alpha, \gamma)^\gamma$.

Case 2: $\gamma < \log^{(\alpha)}(\beta)$. Then $\beta^\gamma < \beta^{\log^{(\alpha)}(\beta)}$. ◀

So inserting $\beta = \log(p_{\max})$, $\gamma = 2^{O(d)}$, we get for any $\alpha \in \mathbb{N}_{>0}$:

$$\log(p_{\max})^{2^{O(d)}} \leq \max\{\text{Tower}(\alpha, 2^{O(d)})^{2^{O(d)}}, \log(p_{\max})^{\log^{(\alpha+1)}(p_{\max})}\}$$

Interestingly enough, $\log^{(\alpha+1)}(p_{\max}) \leq 1$ for $\alpha = 4$, as long as $p_{\max} \leq 2^{65536}$. So one could argue that for any reasonable value of p_{\max} ($\leq 2^{65536}$), the running time $\log(p_{\max})^{2^{O(d)}} \langle I \rangle^{O(1)}$ is FPT w.r.t. d . Of course, it is a bit hypocritical to argue about ‘reasonable’ values (which would be encountered in practice) if at the same time the running time then includes a tower of height 5. Also, the assumption $p_{\max} \leq 2^{65536}$ directly yields an FPT running time. It still seems to us that this is an interesting observation, as this might be as close to an FPT running time as one can get.

In Appendix D, we give a reduction from $Q||\{C_{\max}, C_{\min}\}$ to $P||\{C_{\max}, C_{\min}\}$, producing instances with $d + \tau$ job types, where τ is the number of machine types. This shows that the big open question whether $P||C_{\max}$ is FPT w.r.t. parameter d (see [35]) and the question whether $Q||\{C_{\min}$ is FPT w.r.t. parameters d and τ have the same answer:

► **Theorem 9.** *The following statements are equivalent:*

1. $P||\{C_{\max}, C_{\min}\}$ is FPT w.r.t. parameter d .
2. $Q||\{C_{\max}, C_{\min}\}$ is FPT w.r.t. parameters d and τ .

Related Work

In their paper [18], Goemans and Rothvoss showed the following result:

► **Theorem 10** (Goemans & Rothvoss [18]). *Given rational polyhedra $P, Q \subset \mathbb{R}^N$ where P is bounded, one can find a vector $y \in \text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$ and a vector $x \in \mathbb{Z}_{\geq 0}^{|P \cap \mathbb{Z}^N|}$ such that $y = \sum_{c \in P \cap \mathbb{Z}^N} cx_c$ in time $\langle P \rangle^{2^{O(N)}} \langle Q \rangle^{O(1)}$ or decide that no such y exists. Here, $\langle P \rangle$ and $\langle Q \rangle$ are the encoding lengths of P and Q , respectively.*

They then briefly showed how this can be used directly to solve various scheduling problems. For example, they get an algorithm with running time $(\log(C_{\max}))^{2^{O(d)}} \langle I \rangle^{O(1)}$ for $P||C_{\max}$. The structural result has been generalized by Jansen and Klein:

► **Theorem 11** (Jansen & Klein [18]). *Given rational polyhedra $P, Q \subset \mathbb{R}^N$ where P is bounded, one can find a vector $y \in \text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$ and a vector $x \in \mathbb{Z}_{\geq 0}^{|P \cap \mathbb{Z}^N|}$ such that $y = \sum_{c \in P \cap \mathbb{Z}^N} cx_c$ in time $|V|^{2^{O(N)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)}$ or decide that no such y exists. Here, $\langle P \rangle$ and $\langle Q \rangle$ are the encoding lengths of P and Q , respectively and V is the set of vertices of the integer hull of P .*

Kowalik et al. [32] showed that the doubly-exponential dependency on N is necessary, unless the ETH fails. Knop et al. [30] gave a framework for solving configuration ILPs and depending on the objective, the algorithms have different running times. As part of their framework, they

also generalize the algorithm by Goemans and Rothvoss [18] to multiple types of P-polytopes, which allows modelling uniform and unrelated machines quite efficiently. The right-hand-sides of the systems describing the polytopes (and hence such terms as the makespan or due dates) still appear in the running time in the same way as they do in the algorithm by Goemans and Rothvoss [18]. Knop et al. [29] then show how this framework can be used to obtain parameterized algorithms for various scheduling problems. For identical machines however, their running time that is not parameterized by p_{\max} does not provide an improvement over the running time by Goemans and Rothvoss [18]. When one considers p_{\max} a parameter, there are plenty of other algorithms, mostly using ILPs as a subroutine. Mnich and Wiese [36] use a balancing result in combination with solving an ILP in small dimension. Knop and Koutecký [28] use the normal assignment ILP and solve it with an n -fold algorithm. The framework by Knop et al. [27] uses huge n -fold ILPs and proximity; a similar approach was described by Koutecký and Zink [31]. Govzmann et al. [19] bound the coefficients in the configurations and use an algorithm for ILPs with few constraints.

As mentioned above, Koutecký and Zink [31] gave an algorithm for $P||C_{\max}$ with running time $2^{2^{O(d)}} \|n\|_1^{O(1)}$, brilliantly combining the algorithm by Goemans and Rothvoss [18] with the result by Frank and Tardos [17] to bound the coefficients in terms of $\|n\|_1$ (the total number of jobs). Moreover, they stated in a footnote (without proof) that the result by Goemans and Rothvoss gives a $(\log(p_{\max}))^{f(d)} (\log(\|n\|_1))^{O(1)}$ -time algorithm for $P||C_{\max}$. The running time stated by Goemans and Rothvoss [18] is $(\log(\max\{C_{\max}, \|n\|_1\}))^{f(d)}$ and it is clear that $\|n\|_1$ does not have to be part of the basis, as $\|n\|_1$ only appears in the encoding of Q , but it is not clear at all how the dependency on C_{\max} can be avoided if one uses the algorithm by Goemans and Rothvoss. Using either of our three tools, however, yields a running time like the one claimed by Koutecký and Zink [31]. As mentioned above, Mnich and van Bevern [35] stated the parameterized complexity of $P||C_{\max}$ in high-multiplicity encoding with parameter d as an open problem.

2 Preliminaries

In this section, we introduce the studied problems, notation and terminology. We denote by $[k] := \{1, \dots, k\}$ the numbers up to k and by $v_{\max} := \|v\|_{\infty} = \max_{i=1, \dots, N} \{v_i\}$ the largest (absolute) entry in a vector $v \in \mathbb{R}^N$. As it is custom in parameterized complexity, we sometimes write $f(k)$ to represent *any* computable function that only depends on the parameter k . Hence, it might not be explicitly specified and it might change, e.g. by consuming additional factors that depend only on k . A problem is fixed-parameter tractable (FPT) w.r.t. parameter k if it can be solved in time $f(k) \langle I \rangle^{O(1)}$.

Applications

To show how Theorem 5 and Theorem 7 can be useful, we apply them to various problems, mainly from the area of scheduling, to obtain faster algorithms. Following the notation introduced by Graham et al. [20], we denote scheduling problems by a triple $\alpha|\beta|\gamma$, where α describes the machine setting, β describes a list of additional job constraints and γ describes the objective. In the main part of this paper, we only consider the problems $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$; other problems that appear in Appendix A are described later. For the problem $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$, we are given a job vector $n \in \mathbb{N}_{>0}^d$, a corresponding processing time vector $p \in \mathbb{N}_{>0}^d$ and a number m of identical machines. Formally, the task is to define an assignment $\pi : [d] \mapsto ([m], \llbracket n \rrbracket_1)$ that maps job types to machines (and multiplicities) such that the maximum load $C_{\max} = \max_{i \in [m]} \{L_{\pi, i}\}$ over the machines is

minimized (for objective C_{\max}), the minimum load $C_{\min} = \min_{i \in [m]} \{L_{\pi,i}\}$ is maximized (for objective C_{\min}) or the difference between C_{\max} and C_{\min} is minimized (for objective C_{envy}).² The load $L_{\pi,i}$ of a machine $i \in [m]$ w.r.t. schedule π is defined as the sum of all processing times of the jobs assigned to i , i.e., $L_{\pi,i} := \sum_{\substack{j \in [d] \\ \pi(j)=(i,k)}} kp_j$. Of course, we may not schedule more jobs of a type than we are given. An important concept in scheduling are *configurations*. A configuration is a selection of jobs (represented by a d -dimensional vector) that can be scheduled together on a machine. In the simple case of $P||C_{\max}$, the set of configurations is defined as $\mathcal{C} := \{c \in \mathbb{N}^d \mid p^T c \leq u\}$ when all jobs have to be completed by time u .

As the reader might have noticed, the input to our scheduling problems is given by vectors of dimension d and multiplicities. Classically, in the low-multiplicity encoding, the $\|n\|_1$ jobs would all be listed together with their processing times and other characteristics, even if all jobs were identical. In this paper, all inputs are high-multiplicity encoded. The encoding length of an instance (or similarly for a polytope) I is denoted by $\langle I \rangle$. e.g. for $P||C_{\max}$, we have $\langle I \rangle \leq O(d \log(p_{\max} + n_{\max}) + \log(m))$ as opposed to the classical encoding, which might have size $O(\|n\|_1 \log(p_{\max}))$. Note that one can assume that $m \leq \|n\|_1$, because more than $\|n\|_1$ machines are not necessary, so in the classical encoding, the part that encodes the number of machines vanishes. Depending on the given instance, $\|n\|_1$ might be exponential in $\langle I \rangle$ and m might be as well. This is the reason why the $f(d) \|n\|_1^{O(1)}$ -time algorithm by Koutecký and Zink [31] is not FPT in the high-multiplicity setting.

Note that the scheduling problems described above are optimization problems. However, the framework can only really model decision problems. Fortunately, the optimization problems can be easily solved by solving several decision problems as part of a binary search over some interval $[0, v]$. The key ingredient is that the optimum is always bounded by some value v that depends on the numbers in the input and hence $\log(v) \leq \langle I \rangle^{O(1)}$.³

Throughout this paper, we only consider the decision problems, and as our algorithms all have an $\langle I \rangle^{O(1)}$ -term in the running time anyway, this directly also yields algorithms with the same running time for the optimization problem, except in the C_{envy} -case. There, we get an additional factor, which is p_{\max} . Hence, we will always assume that we are given thresholds ℓ and/or u for the load values when solving these problems. Clearly, (the logarithms of) these are then also part of the encoding. A detailed justification for only considering decision problems is given in Appendix C.

Reducing Coefficients

If the coefficients of an inequality (or equality) are much larger compared to the number of variables and the variables are also bounded by a rather small value, the coefficients can be reduced with the following classical result by Frank and Tardos:

► **Theorem 12** (Frank & Tardos [17]). *For every $w \in \mathbb{R}^N$ and $\Delta \in \mathbb{N}$, there exists a $\bar{w} \in \mathbb{Z}^N$ such that $\|\bar{w}\|_{\infty} \leq (N\Delta)^{O(N^3)}$ and $\text{sign}(w^T x) = \text{sign}(\bar{w}^T x)$ for every $x \in \mathbb{Z}^N$ with $\|x\|_1 \leq \Delta - 1$. Moreover, \bar{w} can be computed in time $N^{O(1)}$.*

► **Corollary 13** (Stated in a similar form by Etscheidt et al. [16]). *For every $w \in \mathbb{R}^N$, $b \in \mathbb{R}$, $\Delta \in \mathbb{N}$, one can compute \bar{w}, \bar{b} with $\|\bar{w}\|_{\infty}, |\bar{b}| \leq (N\Delta)^{O(N^3)}$ in time $N^{O(1)}$ such that for every $x \in [-\Delta, \Delta]^N$, $w^T x \leq b \iff \bar{w}^T x \leq \bar{b}$.*

² Note that for the problems considered in this paper, the order or starting times do not matter; only the load values (and the schedulability of early jobs) are important.

³ Note e.g. in the case of $P||C_{\max}$ that $\log(v) = \log(\sum_{j=1}^d p_j n_j) \leq \log(dp_{\max} n_{\max}) \leq \langle I \rangle^{O(1)}$.

A proof is given in Appendix C. Note that due to the equivalence, zeros are mapped to zeros, negative entries are mapped to negative entries and positive entries are mapped to positive entries. This can be seen by setting x to be a unit vector in Theorem 12.

Another way to greatly reduce a given problem instance (and the set of configurations) is provided by the following result by Govzman et al. [19]. It is the key ingredient used in Appendix B to improve the running time of the additive approximation scheme from [3] in a high-multiplicity setting and can also be used to reduce the coefficients in a PQ-representation. For completeness, we include a proof in Appendix C, as the result has not been published yet. A similar but exponential bound had already been shown by Mnich and Wiese [36].

► **Lemma 14** (Govzmann et al. [19]). *For $P \parallel \{C_{\max}, C_{\min}, C_{\text{envy}}\}$, there exists a kernel where the number of jobs of a specific type on a specific machine is bounded by $2p_{\max}$. So the load of every machine is bounded by $2p_{\max}^2 d$. The kernelization runs in $O(d)$ time.*

PQ-Representations

‘PQ-representations’ are formally defined as follows:

► **Definition 15** (PQ-Representation). *Given a problem X , a PQ-representation of X is a triple (P, Q, m) of polytopes $P, Q \subset \mathbb{R}^N$ and a number m such that for every instance I of X , I is positive if and only if there exists a $y = y^{(1)} + \dots + y^{(m)} \in \text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$ such that for all $i \in [m]$, $y^{(i)} \in P \cap \mathbb{Z}^N$.*

Problems that have a PQ-representation can be solved with the algorithm by Goemans and Rothvoss [18] or the algorithm by Jansen and Klein [23]:

► **Proposition 16** (See [23] and [18]). *A problem with a PQ-representation (P, Q, m) , where $P, Q \subset \mathbb{R}^N$, can be solved in time $|V|^{2^{O(N)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)} (\log(m))^{O(1)}$ or in time $\langle P \rangle^{2^{O(N)}} \langle Q \rangle^{O(1)} (\log(m))^{O(1)}$, where V is the set of vertices of the integer hull of P .*

Bounds on the Vertices of the Integer Hull

Clearly, the number of vertices of the integer hull plays a big role in the running time of Proposition 16. Formally, the integer hull of a polytope $P \subset \mathbb{R}^N$ is defined as the convex hull of $P \cap \mathbb{Z}^N$. There are several upper and lower bounds for the number of vertices of the integer hull, e.g. the one by Berndt et al. [2], which improved the one by Aliev et al. [1]:

► **Theorem 17** (Berndt et al. [2]). *The number of vertices of the integer hull of a polytope $P = \{x \in \mathbb{R}_{\geq 0}^N \mid Ax = b\}$ is bounded by $(NM \log(M\Delta))^{O(M \log(\sqrt{M\Delta}))}$, where $A \in \mathbb{Z}^{M \times N}$ and $\Delta = \|A\|_{\infty}$.*

Note that this does not depend on the right-hand-side but has a stronger dependency on the largest coefficient in the matrix compared to the older result by Hartmann [22] and Cook et al. [7], which appears to be nearly optimal in general [38]:

► **Theorem 18** (Hartmann [22] and Cook et al. [7]). *Let $P = \{x \in \mathbb{R}^N \mid Ax \leq b\}$ be a rational polyhedron with $A \in \mathbb{Z}^{M \times N}$ and let $\Delta = \|A\|_{\infty}$. Then the integer hull of P has at most $(MN\phi)^{O(N)} = (MN \log(\max\{\Delta, \|b\|_{\infty}\}))^{O(N)}$ vertices, where $\phi = O(N \log(\max\{\Delta, \|b\|_{\infty}\}))$ is the maximum encoding length of the inequalities.*

3 Tool 1: The Balancing Result by Govzmann et al.

How Proposition 16 can be useful for solving scheduling and packing problems can be seen at the example of $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$. Remember that we are given load thresholds ℓ and/or u , depending on the objective:

► **Lemma 19.** $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$ has a PQ -representation with m set to the number of machines, $Q_{C_{\max}} = Q_{C_{\min}} = Q_{C_{\text{envy}}} = \left\{ (n_1 \ \dots \ n_d)^T \right\}$ and respectively

$$P_{C_{\max}} = \left\{ c \in \mathbb{R}_{\geq 0}^d \mid p^T c \leq u \right\}, \quad P_{C_{\min}} = \left\{ c \in \mathbb{R}_{\geq 0}^d \mid \begin{pmatrix} p^T \\ -p^T \end{pmatrix} c \leq \begin{pmatrix} \ell + 2p_{\max} \\ -\ell \end{pmatrix} \right\} \quad \text{and}$$

$$P_{C_{\text{envy}}} = \left\{ c \in \mathbb{R}_{\geq 0}^d \mid \begin{pmatrix} p^T \\ -p^T \end{pmatrix} c \leq \begin{pmatrix} u \\ -\ell \end{pmatrix} \right\}.$$

Proof. We only show the claim for $P||C_{\max}$, so let $P := P_{C_{\max}}$ and $Q := Q_{C_{\max}}$. The other proofs work analogously, with the only difference being that we can restrict ourselves to configurations that have load at most $\ell + 2p_{\max}$ in the case of C_{\min} , as we show in Appendix C. Clearly, P and Q are bounded. Consider an instance I of $P||C_{\max}$ and suppose that it is positive, i.e., there is a schedule of all the jobs with makespan at most u . For each machine $k \in [m]$, create a $(d+1)$ -dimensional vector $y^{(k)} := c$ such that c_i is the number of jobs with processing time p_i scheduled on machine k . Clearly, summing up all the vectors $y^{(k)}$ yields $(n_1 \ \dots \ n_d)^T$, since all jobs are scheduled. Moreover, we constructed m such vectors. Since the $y^{(k)}$ are integral, we have $\sum_{k=1}^m y^{(k)} \in \text{int.cone}(P \cap \mathbb{Z}^d) \cap Q$.

Conversely, suppose there is a vector $y \in \text{int.cone}(P \cap \mathbb{Z}^d) \cap Q$. Since $y \in \text{int.cone}(P \cap \mathbb{Z}^d)$, y can be written as a conic combination of integral vectors in P . Let S be the multi-set of these integral vectors, i.e., $\sum_{y^{(k)} \in S} y^{(k)} = y$. Each $y^{(k)} \in S$ corresponds to an integral d -dimensional vector. For each $y^{(k)} \in S$, pick an (idle) machine $k \in [m]$ and for every $i \in [d]$, schedule $y_i^{(k)}$ jobs of type i on k . Since y is a sum of m vectors, we know that $|S| = m$ and since $y \in Q$, $\sum_{y^{(k)} \in S} y_i^{(k)} = n_i$ holds for every $i \in [d]$. So indeed we schedule all the jobs and have enough machines to do so. Since each $y^{(k)} \in S$ also lies in P , the jobs scheduled on the same machine have processing time at most u . So the constructed schedule is feasible and the given instance I is positive. ◀

So a straightforward application of Proposition 16 to the above formulations would yield algorithms running in time $|V|^{2^{O(d)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)}$, where $|V|$ is the number of vertices of P 's integer hull or running time $\langle P \rangle^{2^{O(d)}} \langle Q \rangle^{O(1)}$. Note however that without further consideration, bounds for $|V|$ either contain a term $(\log(p_{\max}))^{O(\log(p_{\max}))}$ or $\log(\ell) \leq \log(u)$, depending on whether we use Theorem 17 or Theorem 18. Moreover, $\langle P \rangle$ also depends on $\log(\ell) \leq \log(u)$. So these do not yet yield a running time $(\log(p_{\max}))^{2^{O(d)}} \langle I \rangle^{O(1)}$. This is where the balancing result by Govzmann et al. (Lemma 14) can help: By paying $O(d)$ time (which vanishes in the other terms), we can assume that $\ell \leq u \leq 2dp_{\max}^2$, as there are at most $2p_{\max}$ jobs of each type on any machine. Then the running time $\langle P \rangle^{2^{O(d)}} \langle Q \rangle^{O(1)}$ becomes

$$(d \log(p_{\max}) + \log(u))^{2^{O(d)}} \langle I \rangle^{O(1)} = (d \log(p_{\max}) + \log(2dp_{\max}^2))^{2^{O(d)}} \langle I \rangle^{O(1)}$$

and with the considerations about optimization (see Appendix C) in mind, we get the following result:

► **Theorem 1.** *The optimization problem $P||\{C_{\max}, C_{\min}\}$ can be solved in time $(d \log(p_{\max}) + \log(2dp_{\max}^2))^{2^{O(d)}} \langle I \rangle^{O(1)}$ and the optimization problem $P||C_{\text{envy}}$ can be solved in time $p_{\max}(d \log(p_{\max}) + \log(2dp_{\max}^2))^{2^{O(d)}} \langle I \rangle^{O(1)}$.*

4 Tool 2: Proximity

In some cases, we might not have such a fine preprocessing algorithm like the one by Govzmann [19] for our problem but we still want to reduce the encoding length of P . In this section, we show how a similar preprocessing can be done for general PQ-representations. Each PQ-representation also has an equivalent ILP and solving a relaxation of this ILP allows us to remove the dependency on the right-hand-side (e.g. u in the case of $P||C_{\max}$):

► **Lemma 20.** *Given a PQ-representation (P, Q, m) with $P = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(P)}x = b^{(P)}\}$ and $Q = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(Q)}x = b^{(Q)}\}$, the points $y \in \text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$ have a one-to-one correspondence to the solutions of the following ILP, projected to block $m + 1$ and the other blocks correspond to vectors $y^{(1)}, \dots, y^{(m)} \in P \cap \mathbb{Z}^N$ such that $y = y^{(1)} + \dots + y^{(m)}$:*

$$\left(\begin{array}{ccc|cc} 0 & 0 & \dots & 0 & A^{(Q)} \\ I & I & \dots & I & -I \\ \hline A^{(P)} & & & & \\ & A^{(P)} & & & \\ & & \ddots & & \\ & & & A^{(P)} & \\ & & & & 0 \end{array} \right) x = \begin{pmatrix} b^{(Q)} \\ \mathbf{0} \\ b^{(P)} \\ b^{(P)} \\ \vdots \\ b^{(P)} \\ \mathbf{0} \end{pmatrix}$$

$x \in \mathbb{Z}_{\geq 0}^{(m+1)N}$

Proof. Suppose we are given a point $y = y^{(1)} + \dots + y^{(m)} \in \text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$ such that for all $i \in [m]$, $y^{(i)} \in P \cap \mathbb{Z}^N$. Then $x := (y^{(1)} \dots y^{(m)} y)^T$ is a solution of the above ILP. Now suppose we have a solution $x = (y^{(1)} \dots y^{(m)} y)^T$. Then by the local, non-negativity and integrality constraints, we have $y^{(i)} \in P \cap \mathbb{Z}^N$ for all $i \in [m]$. By the first set of constraints, we have $y \in Q$ and by the second set of constraints, we have $y = y^{(1)} + \dots + y^{(m)}$. Hence, $y \in \text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$. ◀

The ILP in Lemma 20 has a so-called n -fold structure, as indicated by the framing (see e.g. [13]). We use a proximity result by Cslvjecsek et al. [10] to bound the vectors in P , which allows us to reduce the coefficients in the right-hand-side.⁴ The key idea is to solve a relaxation of the ILP corresponding to the PQ-representation. In general, this *convexified relaxation* of an n -fold ILP looks like this:

$$\begin{aligned} & \max \sum_{i=1}^n (c^{(i)})^T x^{(i)} \\ & \sum_{i=1}^n C^{(i)} x^{(i)} = b^{(0)} \\ & x^{(i)} \in Q^{(i)} := \text{int.hull} \left(\{x \in \mathbb{R}^t \mid B^{(i)}x = b^{(i)}, \mathbf{0} \leq x \leq u\} \right) \quad \forall i \in [n] \end{aligned}$$

where the $C^{(i)} \in \mathbb{Z}^{r \times t}$ are the blocks at the top, $b^{(0)} \in \mathbb{Z}^r$ is the corresponding right-hand-side, the $B^{(i)} \in \mathbb{Z}^{s \times t}$ are the blocks on the diagonal, the $b^{(i)} \in \mathbb{Z}^s$ are the corresponding right-hand-sides, the $c^{(i)} \in \mathbb{Z}^t$ are the parts of the objective function and $u \in \mathbb{Z}^t$ is an upper bound for the variables. Cslvjecsek et al. [10] show that the Lagrangean dual (w.r.t. the

⁴ We cite the first version of their paper here, because it is much easier to see how the approach for solving the relaxation can be adapted to the case where many blocks are identical.

first r constraints) of this problem can be solved with at most $r^{O(1)} \log^2(ntK)$ calls to a separation oracle, where K is the largest number appearing in the objective function, matrix and right-hand-side. The separation oracle has to solve

$$L(\lambda) = \max_{\substack{x^{(i)} \in Q^{(i)} \\ i=1, \dots, n}} \left\{ \sum_{i=1}^n (c^{(i)})^T x^{(i)} + \lambda^T \left(\sum_{i=1}^n C^{(i)} x^{(i)} - b^{(0)} \right) \right\}$$

for a given $\lambda \in \mathbb{R}^r$. In general, this can be done by solving ILPs corresponding to the n blocks $Q^{(i)}$ separately, which can be done in a total time $nt^{O(t)}(st \log(K))^{O(1)}$ using the algorithm by Kannan [26] n times.⁵ However, note that if for two blocks i and j , we have $c^{(i)} = c^{(j)}$, as well as $B^{(i)} = B^{(j)}$ and $b^{(i)} = b^{(j)}$, the ILP has the same optimal solution(s).

When we have computed a solution of the Lagrangean dual, we still have to compute a solution of the primal, i.e., the convexified relaxation. This part is described in more detail in the second version of the paper by Cslovjecsek et al. [9]: While solving the dual with the cutting plane algorithm, each call to the separation oracle yields an optimizer $z^{(j)} \in \mathbb{Z}^{nt}$ of the maximum in iteration j . One then only has to find a vector $x^* \in \mathbb{R}^{nt}$ in the convex hull of the $z^{(j)}$ that also fulfills the linking constraints $\sum_{i=1}^n C^{(i)} x^{(i)} = b^{(0)}$ and maximizes the objective $\max \sum_{i=1}^n (c^{(i)})^T x^{(i)}$. This LP has many variables (equal to the number of calls to the separation oracle) but only $r + 1$ constraints. So it can be solved e.g. with the algorithm by Megiddo [34] (applied to the dual of this LP; it also provides a primal solution) in a time that is polynomial in the number of oracle calls and $2^{O(r^2)}$. Altogether, for τ different blocks, this yields running time

$$\underbrace{r^{O(1)} \log^2(ntK)}_{\text{number of oracle calls}} \underbrace{\tau t^{O(t)} (st \log(K))^{O(1)}}_{\text{time for one oracle call}} + \underbrace{(r^{O(1)} \log^2(ntK))^{O(1)} 2^{O(r^2)}}_{\text{time for solving the LP}} \\ \leq \tau t^{O(t)} 2^{O(r^2)} (sr \log(Kn))^{O(1)}$$

and we get the following result:

► **Proposition 21.** *Let τ be the number of different $(c^{(i)}, B^{(i)}, b^{(i)})$ -pairs, n the number of blocks, r, s, t the block dimensions and K the largest number appearing in the objective function, matrix and right-hand-side. Then the convexified relaxation can be solved in time $\tau t^{O(t)} 2^{O(r^2)} (sr \log(Kn))^{O(1)}$.*

Cslovjecsek et al. [9] show that the convexified relaxation has nice proximity properties, namely they show the following:

► **Theorem 22** (Cslovjecsek et al. [9]). *For every solution x^* of the convexified relaxation, there exists a solution z^* of the corresponding integer program such that $\|x^* - z^*\|_1 \leq (r\Delta(s\Delta))^{O(s)} \Delta^{O(r)}$, where Δ is the largest absolute value in the matrix.*

By first solving the convexified relaxation using Proposition 21 and then applying the proximity bound from Theorem 22, we can bound the solutions of the ILP. Due to Lemma 20, this also bounds the points in P and we obtain the following theorem (for a full proof, see Appendix C):

► **Theorem 23.** *Consider a problem with a PQ-representation (P, Q, m) , where $P = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(P)}x = b^{(P)}\}$ with $A^{(P)} \in \mathbb{Z}^{M^{(P)} \times N}$ and $Q = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(Q)}x = b^{(Q)}\}$ with*

⁵ Cslovjecsek et al. [10] use the algorithm by Eisenbrand and Weismantel [14] instead, but here it makes more sense to use the algorithm by Kannan [26], as our running time will be exponential in t anyway.

$A^{(Q)} \in \mathbb{Z}^{M^{(Q)} \times N}$. Then by solving the convexified relaxation of the corresponding ILP in time $2^{O((M^{(Q)}+N)^2)} (M^{(P)} \log(\max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty, \|b^{(P)}\|_\infty, \|b^{(Q)}\|_\infty\}m))^{O(1)}$ and modifying P , we can assume that the points in P have ℓ_∞ -norm at most

$$((M^{(Q)} + N)M^{(P)} \max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\})^{O((M^{(Q)}+N)M^{(P)})}.$$

We now have all the tools to prove Theorem 5. The key idea is to combine Theorem 23 with the result by Frank and Tardos (Corollary 13). This reduces the coefficients in the inequalities describing P . In particular, it makes their size independent of the original right-hand-side. Again, for a full proof, we refer to Appendix C.

► **Theorem 5.** *If a problem has a PQ-representation (P, Q, m) with $P = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(P)}x = b^{(P)}\}$ and $Q = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(Q)}x = b^{(Q)}\}$ with $A^{(P)} \in \mathbb{Z}^{M^{(P)} \times N}$, $A^{(Q)} \in \mathbb{Z}^{M^{(Q)} \times N}$, $b^{(P)} \in \mathbb{Z}^{M^{(P)}}$ and $b^{(Q)} \in \mathbb{Z}^{M^{(Q)}}$, it can be solved in time*

$$((M^{(P)}M^{(Q)} \log(\max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\}))^{2^{O(N)}} + 2^{O((M^{(Q)})^2)}) (\langle P \rangle \langle Q \rangle \log(m))^{O(1)}.$$

Note how this result also yields an algorithm for $P \parallel \{C_{\max}, C_{\min}, C_{\text{envy}}\}$ with running time $(\log(p_{\max}))^{2^{O(d)}} \langle I \rangle^{O(1)}$, but with worse constants.

5 Tool 3: More Fitting Bounds for the Vertices of the Integer Hull

As mentioned above, given a polytope $P = \{x \in \mathbb{R}^N \mid Ax \leq b\}$, there are upper bounds for the number of vertices of P 's integer hull. But they either depend on $\log(\|b\|_\infty)$ (like Theorem 18) or are exponential in $\log(\|A\|_\infty)$ (like Theorem 17). For using the algorithm by Jansen and Klein [23] – which takes time $|V|^{2^{O(d)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)} \log(m)^{O(1)}$ to solve a PQ-representation – it would be nice to have a bound that does not depend on $\log(\|b\|_\infty)$ but is also not exponential in $\log(\|A\|_\infty)$. Fortunately, the proof by Berndt et al. [2] can be adapted to get the following result:

► **Theorem 6.** *The number of vertices of the integer hull of a polytope $P = \{x \in \mathbb{R}_{\geq 0}^N \mid Ax = b\}$ is bounded by $N^M O(M \log(M\Delta))^N$, where $A \in \mathbb{Z}^{M \times N}$ and $\Delta = \|A\|_\infty$.*

Proof. (Sketch). Denote by P_I the integer hull of P . It is well-known that P has at most $\binom{N}{M}$ vertices (the basic feasible solutions). Eisenbrand and Weismantel [14] have shown that for each vertex z^* of P_I , there exists a vertex x^* of P such that $\|x^* - z^*\|_1 \leq M(2M\Delta + 1)^M$.⁶ The idea of the proof by Berndt et al. [2] is to use this proximity result to remove the dependency on $\|b\|_\infty$. Each vertex of P_I is associated with at least one vertex of P ; if we were to draw balls around each vertex of P with the radius set to the proximity bound, each vertex of P_I would be in at least one of these circles. Instead of rasterizing the whole P_I , Berndt et al. only rasterize the regions around the vertices of P . This yields at most $O(\log(M(2M\Delta + 1)^M))^N = O(M \log(M\Delta))^N$ boxes per vertex of P , each of which contains at most one vertex of P_I . So P_I has at most

$$\binom{N}{M} O(M \log(M\Delta))^N \leq N^M O(M \log(M\Delta))^N$$

vertices. Berndt et al. then proceed to reduce the exponent so that it does not depend on N , which might in some cases be rather large compared to Δ and M . ◀

⁶ To be precise, they showed that for each optimal solution of the associated LP, there exists an optimal solution of the corresponding ILP with this distance. But by perturbing the linear objective function, every vertex of P_I can be made a unique optimum.

We can use this result to solve PQ-representations with the algorithm by Jansen and Klein [23]:

► **Theorem 7.** *If a problem has a PQ-representation (P, Q, m) with $P = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(P)}x = b^{(P)}\}$ and $Q = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(Q)}x = b^{(Q)}\}$ with $A^{(P)} \in \mathbb{Z}^{M^{(P)} \times N}$, $A^{(Q)} \in \mathbb{Z}^{M^{(Q)} \times N}$, $b^{(P)} \in \mathbb{Z}^{M^{(P)}}$ and $b^{(Q)} \in \mathbb{Z}^{M^{(Q)}}$, it can be solved in time*

$$(N^{M^{(P)}} (M^{(P)} \log(\|A^{(P)}\|_{\infty}))^{2^{O(N)}} (\langle P \rangle \langle Q \rangle \log(m))^{O(1)}).$$

Proof. We aim to solve (P, Q, m) with the $|V|^{2^{O(N)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)} \log(m)^{O(1)}$ -time algorithm by Jansen and Klein [23] (see Proposition 16), so we need to bound $|V|$, the number of vertices of P 's integer hull. For this, we use Theorem 6, which yields

$$|V| \leq N^{M^{(P)}} O(M^{(P)} \log(M^{(P)} \|A^{(P)}\|_{\infty}))^N.$$

In total, we get the following running time:

$$\begin{aligned} & |V|^{2^{O(N)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)} \log(m)^{O(1)} \\ &= (N^{M^{(P)}} O(M^{(P)} \log(M^{(P)} \|A^{(P)}\|_{\infty}))^N)^{2^{O(N)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)} \log(m)^{O(1)} \\ &= (N^{M^{(P)}} (M^{(P)} \log(\|A^{(P)}\|_{\infty}))^{2^{O(N)}} (\langle P \rangle \langle Q \rangle \log(m))^{O(1)} \end{aligned}$$

◀

Observe that this also yields an algorithm for $P \parallel \{C_{\max}, C_{\min}, C_{\text{envy}}\}$ with running time $(\log(p_{\max}))^{2^{O(d)}} \langle I \rangle^{O(1)}$ (again, with worse constants than in Theorem 1).

6 Conclusion and Future Work

In this work, we investigated three tools that improve the running time of the algorithm by Goemans and Rothvoss [18]. The first tool could be considered a problem-specific preprocessing, the second tool uses a special relaxation of an n -fold ILP and proximity and the third tool utilizes the framework by Jansen and Klein [23] together with new customized bounds for the vertices of the integer hull. Overall, each tool provides a $(\log(p_{\max}))^{2^{O(d)}} \langle I \rangle^{O(1)}$ -time algorithm for $P \parallel \{C_{\max}, C_{\min}, C_{\text{envy}}\}$, which is FPT w.r.t. d if the processing times are given in unary. The second and third tool also yield similar algorithms for other problems. The second tool provides a bound for the variables in P . In some cases, there are natural bounds for the variables in P already, e.g. $\|n\|_1$ for $Q \parallel \{C_{\max}, C_{\min}, C_{\text{envy}}\}$. Of course, one can also use such bounds instead of proximity, if they produce a better result. On another note, it would be interesting to apply the improved framework to other scheduling and packing problems like the ones considered by Goemans and Rothvoss [18] and to see how eliminating the right-hand-side influences the running times. Another very interesting direction is incorporating our ideas into the framework by Knop et al. [30] to handle multiple types of P -polytopes. This might then extend our results to scheduling on uniform and unrelated machines with and without deadlines or release dates, parallel task scheduling, 2-dimensional packing as well as other objectives in an efficient way.

References

- 1 I. Aliev, J. A. De Loera, F. Eisenbrand, T. Oertel, and R. Weismantel. The support of integer optimal solutions. *SIAM Journal on Optimization*, 28(3):2152–2157, 2018. doi: 10.1137/17M1162792.

- 2 Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. New bounds for the vertices of the integer hull. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 25–36. SIAM, 2021. doi:10.1137/1.9781611976496.3.
- 3 Moritz Buchem, Lars Rohwedder, Tjark Vredeveld, and Andreas Wiese. Additive approximation schemes for load balancing problems. *CoRR*, abs/2007.09333, 2020. arXiv:2007.09333.
- 4 Moritz Buchem, Lars Rohwedder, Tjark Vredeveld, and Andreas Wiese. Additive approximation schemes for load balancing problems. In *ICALP*, volume 198 of *LIPICs*, pages 42:1–42:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.42.
- 5 Lin Chen, Klaus Jansen, Wenchang Luo, and Guochuan Zhang. An efficient PTAS for parallel machine scheduling with capacity constraints. In *COCOA*, volume 10043 of *Lecture Notes in Computer Science*, pages 608–623. Springer, 2016. doi:10.1007/978-3-319-48749-6_44.
- 6 Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *Journal of Computer and System Sciences*, 96:1–32, 2018. doi:doi.org/10.1016/j.jcss.2018.03.005.
- 7 W. Cook, M. Hartmann, R. Kannan, and C. McDiarmid. On integer points in polyhedra. *Combinatorica*, 12(1):27–37, March 1992. doi:10.1007/BF01191202.
- 8 Gabriel Cramer. *Introduction à l'analyse des lignes courbes algébriques*. chez les frères Cramer et C. Philibert, 1750. doi:10.3931/e-rara-4048.
- 9 Jana Cslovjecssek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-structured integer and linear programming in strongly polynomial and near linear time. In *SODA*, pages 1666–1681. SIAM, 2021. doi:10.1137/1.9781611976465.101.
- 10 Jana Cslovjecssek, Friedrich Eisenbrand, and Robert Weismantel. N-fold integer programming via LP rounding. *CoRR*, abs/2002.07745, 2020. arXiv:2002.07745.
- 11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015. doi:10.5555/2815661.
- 12 Max A. Deppert and Klaus Jansen. Near-linear approximation algorithms for scheduling problems with batch setup times. In *SPAA*, pages 155–164. ACM, 2019. doi:10.1145/3323165.3323200.
- 13 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. In *ICALP*, volume 107 of *LIPICs*, pages 49:1–49:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.49.
- 14 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. *ACM Transactions on Algorithms*, 16(1), November 2019. Place: New York, NY, USA Publisher: Association for Computing Machinery. doi:10.1145/3340322.
- 15 Leah Epstein and Asaf Levin. Minimum weighted sum bin packing. In *WAOA*, volume 4927 of *Lecture Notes in Computer Science*, pages 218–231. Springer, 2007. doi:10.1007/978-3-540-77918-6_18.
- 16 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. *Journal of Computer and System Sciences*, 84:1–10, 2017. doi:10.1016/j.jcss.2016.06.004.
- 17 A. Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, jan 1987. doi:10.1007/BF02579200.
- 18 Michel X. Goemans and Thomas Rothvoss. Polynomiality for bin packing with a constant number of item types. *J. ACM*, 67(6), nov 2020. doi:10.1145/3421750.
- 19 A. Govzmann, M. Mnich, and S. Omlor. Faster algorithms for parallel and related machine scheduling, 2023. Manuscript.
- 20 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. In P.L. Hammer, E.L. Johnson,

- and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979. doi:10.1016/S0167-5060(08)70356-X.
- 21 Jacques Hadamard. Resolution d’une question relative aux déterminants. *Bull. des Sciences Math.*, 2:240–246, 1893. URL: <https://ci.nii.ac.jp/naid/20000814080/en/>.
 - 22 Mark Evan Hartmann. *Cutting Planes and the Complexity of the Integer Hull*. PhD thesis, Cornell University, 1989.
 - 23 Klaus Jansen and Kim-Manuel Klein. About the structure of the integer cone and its application to bin packing. *Mathematics of Operations Research*, 45(4):1498–1511, 2020. doi:10.1287/moor.2019.1040.
 - 24 Klaus Jansen, Alexandra Lassota, and Marten Maack. Approximation algorithms for scheduling with class constraints. In *SPAA*, pages 349–357. ACM, 2020. doi:10.1145/3350755.3400247.
 - 25 Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. Near-linear time algorithm for n -fold ilps via color coding. *SIAM Journal on Discrete Mathematics*, 34(4):2282–2299, 2020. doi:10.1137/19M1303873.
 - 26 Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987. URL: <http://www.jstor.org/stable/3689974>.
 - 27 Dušan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. High-multiplicity n -fold ip via configuration lp. *Mathematical Programming*, 200(1):199–227, Jun 2023. doi:10.1007/s10107-022-01882-9.
 - 28 Dušan Knop and Martin Koutecký. Scheduling meets n -fold integer programming. *J. of Scheduling*, 21(5):493–503, oct 2018. doi:10.1007/s10951-017-0550-0.
 - 29 Dušan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Multitype integer monoid optimization and applications, 2019. arXiv:1909.07326.
 - 30 Dušan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Parameterized complexity of configuration integer programs. *Operations Research Letters*, 49(6):908–913, 2021. doi:10.1016/j.orl.2021.11.005.
 - 31 Martin Koutecký and Johannes Zink. Complexity of scheduling few types of jobs on related and unrelated machines. In *ISAAC*, volume 181 of *LIPICs*, pages 18:1–18:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ISAAC.2020.18.
 - 32 Lukasz Kowalik, Alexandra Lassota, Konrad Majewski, Michal Pilipczuk, and Marek Sokolowski. Detecting points in integer cones of polytopes is double-exponentially hard. In *SOSA*, pages 279–285. SIAM, 2024.
 - 33 Jan Karel Lenstra and David B. Shmoys. Elements of scheduling, 2020. arXiv:2001.06005.
 - 34 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, jan 1984. doi:10.1145/2422.322418.
 - 35 Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, dec 2018. doi:10.1016/j.cor.2018.07.020.
 - 36 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1–2):533–562, December 2015. Place: Berlin, Heidelberg Publisher: Springer-Verlag. doi:10.1007/s10107-014-0830-9.
 - 37 Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.
 - 38 Nikolai Yu. Zolotykh. On the number of vertices in integer linear programming problems, 2006. arXiv:math/0611356.

A Applications

In this section, we apply Theorem 5 to $P|d_j, \text{vec}, \text{class}, \text{cap}| \{C_{\max}, C_{\min}, C_{\text{envy}}\}$ (a quite general problem with interesting special cases), the objective $\sum w_j U_j$, setup times, MINSUM-WEIGHTED-BINPACKING and uniform n -fold ILPs.

The other objective we consider in this section is minimizing the weighted number of late jobs ($\sum w_j U_j$). Each job has a due date d_j and a weight w_j and the goal is to find a schedule such that the weight of the jobs missing their due date is minimized. Since it does not matter by how much late jobs miss their due date, they might as well not be scheduled at all (or way after the other jobs) and we may focus on finding a set of jobs that can be scheduled early. For those jobs, it then only matters on which machine they are scheduled, not in which order, as it is well-known that they may be scheduled in EDD (earliest due date) order [33].

We also consider several job characteristics: Deadlines (d_j), multidimensional jobs (vec), class constraints (class), setup times (s_j) and capacity constraints (cap). If there are deadlines (not to be confused with the due dates from the $\sum w_j U_j$ -objective), each job has a deadline d_j and may not finish after it. In the vec-variant (also referred to as *vector scheduling*), the processing times $p_j \in \mathbb{N}_{>0}^M$ are M -dimensional vectors. If class constraints are present, each job has a class $i \in [K_\#]$, we are given a number $C^{\text{class}} \in \mathbb{N}_{>0}$ and there may only be jobs of at most C^{class} different classes scheduled on each machine. This constraint has e.g. been studied in [24]. Similarly, with setup times, each job has a class and starting a job from a new class invokes a setup time s_j (see e.g. [12]). The capacity constraints just forbid more than $C^{\text{cap}} \in \mathbb{N}_{>0}$ jobs to be scheduled on any machine. This has been studied in [5].

Note that we can always apply Lemma 8 to the running times stated in the theorems of this section, like we did in Section 1 for $P \parallel \{C_{\max}, C_{\min}, C_{\text{envy}}\}$.

We can simplify the running time of Theorem 5 and Theorem 7. Usually, N should be the largest of the three matrix dimensions, anyway:

► **Corollary 24.** *If a problem has a PQ-representation (P, Q, m) with $P = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(P)}x = b^{(P)}\}$ and $Q = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(Q)}x = b^{(Q)}\}$ with $A^{(P)} \in \mathbb{Z}^{M^{(P)} \times N}$, $A^{(Q)} \in \mathbb{Z}^{M^{(Q)} \times N}$, $b^{(P)} \in \mathbb{Z}^{M^{(P)}}$ and $b^{(Q)} \in \mathbb{Z}^{M^{(Q)}}$, it can be solved in time $(\log(\|A^{(P)}\|_\infty))^{2^{O(K)}} (\langle P \rangle \langle Q \rangle \log(m))^{O(1)}$, where $K := \max\{N, M^{(P)}, M^{(Q)}\}$.*

A.1 A Quite General Scheduling Problem

In $P \mid d_j, \text{vec}, \text{class}, \text{cap} \parallel \{C_{\max}, C_{\min}, C_{\text{envy}}\}$ we are given either a lower bound ℓ (C_{\min}), an upper bound u for the load of each machine (C_{\max}), or both (C_{envy}). So the goal is to schedule all $\|n\|_1$ jobs on the m machines while satisfying the machine load bounds and several other constraints:

- Jobs are M -dimensional, i.e., their processing times p_j are M -dimensional vectors, as are the lower and upper bounds on the machine loads,
- each job j has a deadline d_j (which is *not* an M -dimensional vector; otherwise, we might not be able to order jobs by their due date) it has to be finished before,
- each job has a class $i \in [K_\#]$ and there may at most be C^{class} different classes represented on each machine and
- each machine can only schedule C^{cap} jobs.

So the input consists of the load bounds ℓ and u (depending on the objective), the number of machines, the capacities C^{class} and C^{cap} and finally the information about the jobs: For each distinct tuple of deadline, processing time vector and class, we have a multiplicity n_i . In this context, the number of job types d refers to the number of these distinct tuples. So d might be as large as $d_\# p_{\max}^M K_\#$, where $d_\#$ is the number of different deadlines, p_{\max} is the largest entry in all the processing time vectors, M is the dimension of the processing time vectors and $K_\#$ is the number of classes. We assume that the deadlines are ordered non-decreasingly, i.e., d_1 is the smallest deadline.

► **Lemma 25.** $P|d_j, \text{vec}, \text{class}, \text{cap}| \{C_{\max}, C_{\min}, C_{\text{envy}}\}$ has a PQ-representation.

Proof. Set m to the number of machines. Let $N := d + K_{\#} + K_{\#} + (2 + d_{\#}M + 2M + 3K_{\#})$ be the dimension of P , which is defined as

$$P = \left\{ \begin{pmatrix} c \\ x \\ z \\ s \end{pmatrix} \in \mathbb{R}_{\geq 0}^N \mid A \begin{pmatrix} c \\ x \\ z \\ s \end{pmatrix} = b \right\} \quad \text{and let } Q = \left\{ (n \ x \ z \ s)^T \right\}.$$

We describe A and b by a system of equalities and inequalities. The part $s \in \mathbb{R}_{\geq 0}^{2+d_{\#}M+2M+3K_{\#}}$ of the vectors in P are slack variables that we will not explicitly write down for the sake of visibility. Just note that there are $2 + d_{\#}M + 2M + 3K_{\#}$ constraints that have inequalities instead of equalities.

$$\begin{aligned} \mathbf{1}^T c &\leq C^{\text{cap}} && \text{(capacity)} \\ \sum_{k=1}^j (p_{k,i})^T c^{(k)} &\leq d_j && \forall j \in [d_{\#}], i \in [M] \quad \text{(deadlines)} \\ \mathbf{1}^T c^{(i)} &= x_i && \forall i \in [K_{\#}] \quad \text{(class)} \\ x_i \|n\|_1 &\geq z_i && \forall i \in [K_{\#}] \quad \text{(class)} \\ z_i \|n\|_1 &\geq x_i && \forall i \in [K_{\#}] \quad \text{(class)} \\ z_i &\leq 1 && \forall i \in [K_{\#}] \quad \text{(class)} \\ \sum_{i=1}^{K_{\#}} z_i &\leq C^{\text{class}} && \text{(class)} \\ p_i^T c &\leq \left\{ u_i, \sum_{j=1}^d p_{j,i} \right\} && \forall i \in [M] \quad \text{(objective)} \\ p_i^T c &\geq \{\ell_i, 0\} && \forall i \in [M] \quad \text{(objective)} \end{aligned}$$

Here – slightly abusing notation – $c^{(j)}$ is the sub-vector of c that corresponds to jobs that have due date d_j , while $c^{(i)}$ is the sub-vector of c that corresponds to jobs that have class i . The processing time matrix corresponding to the jobs of due date d_j is denoted by p_j and $p_{j,i}$ refers to the processing times of the jobs with due date j in dimension i . The actual values in the objective constraints depend on the objective function: For C_{\max} , we have $0 \leq p_i^T c \leq u_i$; for C_{\min} , we have $\ell_i \leq p_i^T c \leq \sum_{j=1}^d p_{j,i}$ and for C_{envy} , we have $\ell_i \leq p_i^T c \leq u_i$.

To prove that (P, Q, m) forms a PQ-representation for our problem, we first argue that we have $y = (c \ x \ z \ s)^T \in P \cap \mathbb{Z}^N$ if and only if c is a valid configuration. The rest of the proof is then easy.

To this end, suppose that we have a point $y = (c \ x \ z \ s)^T \in P \cap \mathbb{Z}^N$. First, observe that due to the objective constraints, w.r.t. the load, c is a valid configuration for either of the three objectives. Clearly, c fulfills the capacity constraints. The deadline constraints $\sum_{k=1}^j (p_{k,i})^T c^{(k)} \leq d_j$ assure that for each deadline, the jobs selected by c that have this (or an earlier) deadline can be finished before the deadline, in each dimension. This constraint implicitly uses the fact that if a set of jobs can be scheduled on a machine without violating any due date, then they can be scheduled in EDD-order (i.e., in non-decreasing order of due dates). The class constraints are slightly more tricky: The first constraints set the value x_i to the number of jobs selected by c that belong to class i . The second, third and fourth groups

of class constraints assure that whenever $x_i > 0$, then $z_i = 1$ and otherwise $z_i = 0$: The fourth constraint just means that z_i is either 0 or 1. If $x_i = 0$, the second constraint forces $z_i = 0$ and the third constraint is irrelevant. If $x_i > 0$, the second constraint is irrelevant and the third one means that z_i cannot be zero (and must hence be 1). The fifth class constraint now assures that the number of classes where $x_i > 0$ is at most C^{class} . So c fulfills the class constraints and hence is a valid configuration of the given problem.

If we – on the other hand – are given a valid configuration, we can simply create a vector $y = (c \ x \ z \ s)^T$ by

- setting the entries in c according to the number of jobs of each type appearing in the configuration,
- setting x_i to the number of jobs of class i that are in the configuration for all $i \in [K_\#]$,
- setting $z_i = 1$ if $x_i > 0$ and $z_i = 0$ otherwise for all $i \in [K_\#]$ and
- setting the s -entries to the slack of the corresponding inequalities.

By the above arguments, $y \in P \cap \mathbb{Z}^N$.

Now assume that there is a point $y = \sum_{i=1}^m y^{(i)} \in \text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$ that is the sum of m vectors $y^{(i)} \in P \cap \mathbb{Z}^N$. By the above observations, each $y^{(i)}$ corresponds to a set of jobs that may be scheduled together on a single machine. The number of configurations matches the number of machines. Moreover, the jobs in the configurations sum up to all the jobs in the scheduling instance (since the c -vectors sum up to the job vector n). So the vector y gives rise to a feasible schedule.

In the other direction, a schedule gives us m configurations, each of which can be modelled as a vector $y^{(i)} \in P \cap \mathbb{Z}^N$. The job part sums up to n . So the sum of these vectors is a point in $\text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$.

It remains to show that P is bounded. Due to the objective constraints, c is bounded. Moreover, x and z are bounded because of the class constraints. If we add slack variables s , these are also bounded. ◀

► **Theorem 26.** $P|d_j, \text{vec}, \text{class}, \text{cap}| \{C_{\max}, C_{\min}, C_{\text{envy}}\}$ can be solved in time

$$(\log(\max\{\|n\|_1, p_{\max}\}))^{2^{O(d+Md_\#)}} \langle I \rangle^{O(1)},$$

where the $\|n\|_1$ term is only in the basis due to the class constraints.

Proof. Consider the constraints describing P . The parameters in our PQ-representation in Lemma 25 are: $K = \max\{M^{(P)}, M^{(Q)}, N\} = O(d + K_\# + Md_\#) = O(d + Md_\#)$ (the number of job types d is at least as large as the number of classes $K_\#$), $\|A^{(P)}\|_\infty = \max\{\|n\|_1, p_{\max}\}$ and $\langle P \rangle, \langle Q \rangle, \log(m) = \langle I \rangle^{O(1)}$. Note that $\|n\|_1$ only appears in the class constraints.

Corollary 24 yields running time:

$$\begin{aligned} & (\log(\|A^{(P)}\|_\infty))^{2^{O(K)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)} \log(m)^{O(1)} \\ &= (\log(\max\{\|n\|_1, p_{\max}\}))^{2^{O(d+Md_\#)}} \langle I \rangle^{O(1)} \end{aligned}$$

◀

As noted above, the problem $P|d_j, \text{vec}, \text{class}, \text{cap}| \{C_{\max}, C_{\min}, C_{\text{envy}}\}$ is quite general, so it captures many special cases, for some of which we now give corollaries of Theorem 26:

► **Corollary 27.** $P|\text{class}| \{C_{\max}, C_{\min}, C_{\text{envy}}\}$ can be solved in time

$$(\log(\max\{\|n\|_1, p_{\max}\}))^{2^{O(d)}} \langle I \rangle^{O(1)}.$$



► **Corollary 28.** $Q|\text{cap}|\{C_{\max}, C_{\min}, C_{\text{envy}}\}$ can be solved in time

$$(\log(p_{\max}))^{2^{O(d)}} \langle I \rangle^{O(1)}.$$

► **Corollary 29.** $P|\text{vec}|\{C_{\max}, C_{\min}, C_{\text{envy}}\}$ can be solved in time

$$(\log(p_{\max}))^{2^{O(d+M)}} \langle I \rangle^{O(1)}.$$

A.2 Setup Times

Note that in the problem $P|s_j|C_{\max}, C_{\min}, C_{\text{envy}}\}$, the order or starting time of jobs might matter in general, as one can increase the load of a machine by alternating between jobs of different classes to meet some C_{\min} or C_{envy} requirement.⁷ In the following, we assume that no such unnecessary setups are paid (i.e., jobs of the same class are scheduled in a batch). So we assume that the setup s_j of a class is only paid once per machine. In the context of $P|s_j|C_{\max}, C_{\min}, C_{\text{envy}}\}$, the number d of job types might be as large as the number of different processing times multiplied by the number of classes $s_{\#}$.

► **Corollary 30.** $P|s_j|\{C_{\max}, C_{\min}, C_{\text{envy}}\}$ has a PQ-representation, assuming that each class induces only one setup per machine.

Proof. We do not go into too much detail here, as the proof is quite similar to the one of Lemma 25. The vectors in P have a configuration part c , a 1 and parts x and z for the setups. These parts fulfill a similar role as in the class constraints: For a class $i \in [s_{\#}]$, the variable x_i is set to the number of jobs from class i that are in the configuration c . Then if $x_i > 0$, z_i is set to 1 and otherwise $z_i = 0$. This works with the same constraints as in Lemma 25. But instead of forcing $z_i \leq 1$ for all i , we change the objective constraints to:

$$\{\ell_i, 0\} \leq p_i^T c + \sum_{j=1}^{s_{\#}} s_j z_j \leq \left\{ u_i, \sum_{j=1}^d p_j \right\} \quad \forall i \in [M]$$

This way, the setups induced by a class being represented on a machine are included in the load bounds. ◀

► **Corollary 31.** $P|s_j|\{C_{\max}, C_{\min}, C_{\text{envy}}\}$ can be solved in time

$$\log(\max\{\|n\|_1, p_{\max}, s_{\max}\})^{2^{O(d)}} \langle I \rangle^{O(1)},$$

assuming that each class induces only one setup per machine.

Proof. Again, the proof is similar to the one of Theorem 26. Just note that the setups s_i appear in the matrix $A^{(P)}$. There are $O(d + s_{\#}) = O(d)$ variables, so calculations similar to the ones in the proof of Theorem 26 yield the claimed running time. ◀

A.3 Weighted Number of Late Jobs

We now consider the objective $\sum w_j U_j$. In this context, we are only concerned with the early jobs (those that make their due date) and we can assume that these early jobs are scheduled in EDD order (i.e. in order of non-decreasing due dates) on each machine. The late jobs may

⁷ Of course, for the objective C_{\max} , it would not make sense to pay extra setups in an optimal solution.

all be scheduled afterwards on an arbitrary machine. Note that in this problem, a job j has an M -dimensional processing time p_j , a (one-dimensional) due date d_j and a weight w_j . So our parameter d (the number of different job types) naturally extends to the number of different (p_j, d_j, w_j) -tuples. So if we denote the number of different processing times, due dates and weights by $p_\#, d_\#$ and $w_\#$, respectively, d may be as large as $p_\# d_\# w_\# \leq p_{\max}^M d_{\max} w_{\max}$. We are given a threshold T for the total weight of the late jobs.

► **Lemma 32.** *The problem $P|\text{vec}|\sum w_j U_j$ has a PQ-representation.*

Proof. Set m to the number of machines,

$$P = \left\{ \begin{pmatrix} c \\ x \\ s \end{pmatrix} \in \mathbb{R}_{\geq 0}^{d+1+d_\#M} \mid A \begin{pmatrix} c \\ x \\ s \end{pmatrix} = b \right\}$$

and

$$Q = \left\{ (\tilde{n}_1 \ \dots \ \tilde{n}_d \ x \ s)^T \mid \sum_{i=1}^d w_i - T \leq x, \forall i \in [d] : \mathbf{0} \leq \tilde{n}_i \leq n_i \right\},$$

where A and b are given by the following constraints:

$$\begin{aligned} \sum_{k=1}^j (p_{k,i})^T c^{(k)} &\leq d_j & \forall j \in [d_\#], i \in [M] & \quad (\text{due dates}) \\ w^T c &= x & & \quad (\text{objective}) \end{aligned}$$

Here, for all $j \in [d_\#]$, p_j is a matrix containing the vectors of processing times that correspond to the jobs of the j -th due date. Then $p_{j,i}$ contains the processing times in dimension i corresponding to the jobs of due date j . Similarly, $c^{(j)}$ includes the job types that have due date d_j .

As we showed similarly in the proof of Lemma 25, the c -part of the integral vectors in P are valid configurations. In this case, ‘valid’ means that the jobs in c can be scheduled on the same machine (in EDD order) without missing any due dates (in any of the M dimensions). The objective constraint sets x to the total weight of all the jobs selected by c . The due date constraints make sure that no job misses its due date in any dimension if they are scheduled in EDD order. Given a threshold T for the desired weight, a solution of the scheduling problem boils down to a subset of jobs that can be scheduled early and their assignment to the machines such that their total weight is at least $\sum_{i=1}^d w_i - T$ (and hence the weight of all the late jobs is at most T). This is exactly what a point in $\text{int.cone}(P \cap \mathbb{Z}^{d+1+d_\#M}) \cap Q$ corresponds to: A set of configurations (containing the early jobs, hence the values \tilde{n}_i instead of n_i in the description of Q) that have total weight at least $\sum_{i=1}^d w_i - T$. ◀

We can now give a parameterized algorithm for $P|\sum w_j U_j$:

► **Theorem 33.** *The problem $P|\text{vec}|\sum w_j U_j$ can be solved in time*

$$(\log(\max\{p_{\max}, w_{\max}\}))^{2^{O(d+d_\#M)}} \langle I \rangle^{O(1)}.$$

Proof. The parameters in our PQ-representation in Lemma 32 are: $\langle P \rangle, \langle Q \rangle, \log(m) = \langle I \rangle^{O(1)}$, $\|A^{(P)}\|_\infty = \max\{p_{\max}, w_{\max}\}$ and $K = \max\{M^{(P)}, M^{(Q)}, N\} = O(d + d_\#M)$. Then Corollary 24 yields running time:

$$\begin{aligned} &(\log(\|A^{(P)}\|_\infty))^{2^{O(K)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)} \log(m)^{O(1)} \\ &= (\log(\max\{p_{\max}, w_{\max}\}))^{2^{O(d+d_\#M)}} \langle I \rangle^{O(1)} \end{aligned}$$

► **Corollary 34.** *The problem $P \parallel \sum w_j U_j$ can be solved in time*

$$(\log(\max\{p_{\max}, w_{\max}\}))^{2^{O(d)}} \langle I \rangle^{O(1)}.$$

Note that $d_{\#} \leq d$.

A.4 Uniform N-Fold ILPs

Theorem 5 can also be used to solve uniform n -fold ILPs. In general, these ILPs have the following form:

$$\begin{aligned} & \max \sum_{i=1}^n c^{(i)} x^{(i)} \\ & \begin{pmatrix} C^{(1)} & \cdots & C^{(n)} \\ B^{(1)} & & \\ & \ddots & \\ & & B^{(n)} \end{pmatrix} \begin{pmatrix} x^{(1)} \\ \vdots \\ x^{(n)} \end{pmatrix} = \begin{pmatrix} b^{(0)} \\ b^{(1)} \\ \vdots \\ b^{(1)} \end{pmatrix} \\ & \mathbf{0} \leq x^{(i)} \leq u^{(i)} \quad \forall i \in [n] \\ & x^{(i)} \in \mathbb{Z}^t \quad \forall i \in [n] \end{aligned}$$

where $C^{(i)} \in \mathbb{Z}^{r \times t}$, $B^{(i)} \in \mathbb{Z}^{s \times t}$, $b^{(0)} \in \mathbb{Z}^r$, $b^{(1)} \in \mathbb{Z}^s$, $c \in \mathbb{Z}^t$ and $u^{(i)} \in (\mathbb{Z}_{>0} \cup \{\infty\})^t$. Here, we consider the special case with $C^{(1)} = \dots = C^{(n)}$, $B^{(1)} = \dots = B^{(n)}$, $b^{(1)} = \dots = b^{(n)}$, $c^{(1)} = \dots = c^{(n)}$ and $u^{(1)} = \dots = u^{(n)}$. By standard bounds for the solutions of (I)LPs, we know that the objective value is bounded by some U that has an encoding length bounded by $\langle I \rangle^{O(1)}$ (see Schrijver [37] for bounds on the ℓ_{∞} -norm of (I)LP solutions; one could even use proximity arguments like in [25]), if the bounds u were infinite in some directions. So by doing a binary search over the objective values, we can assume that we are looking for a solution x with $\sum_{i=1}^n c x^{(i)} \geq T$ for some $T \in [U]$. If the ILP is bounded, the objective value is also bounded and doing a binary search over the possible objective values only adds a factor of $\langle I \rangle^{O(1)}$.

► **Lemma 35.** *The uniform n -fold ILP has a PQ-representation with $m = n$,*

$$P = \left\{ \begin{pmatrix} x \\ s \\ g \\ v \end{pmatrix} \in \mathbb{R}_{\geq 0}^{3t+1} \mid \begin{pmatrix} B & 0 & 0 \\ I & I & 0 \\ C & 0 & -I \\ c^T & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ s \\ g \end{pmatrix} = \begin{pmatrix} b^{(1)} \\ u \\ \mathbf{0} \\ 0 \end{pmatrix} \right\}$$

and

$$Q = \left\{ (x \ s \ b^{(0)} \ v)^T \in \mathbb{R}^{3t+1} \mid T \leq v \right\}.$$

Proof. Suppose the given uniform n -fold ILP has a solution $x^* \in \mathbb{Z}^{nt}$ with objective value $\sum_{i=1}^n c^T (x^*)^{(i)} \geq T$. Then each of the n bricks $(x^*)^{(i)} \in \mathbb{Z}^t$ can be augmented to a vector $(x \ s \ g \ v)^T \in P$ by setting the value $v = c^T (x^*)^{(i)}$, the solution $x = (x^*)^{(i)}$, the slack $s = u - (x^*)^{(i)}$ and the global part $g = C(x^*)^{(i)}$. The sum of all these augmented vectors is in Q : The values for x and s are unconstrained; we have $\sum_{i=1}^n C(x^*)^{(i)} = b^{(0)}$, which is exactly the sum of all the g -entries; the value is $\sum_{i=1}^n c^T (x^*)^{(i)}$, which is at least T by assumption. So $\text{int.cone}(P \cap \mathbb{Z}^{3t+1}) \cap Q$ is not empty.

Now suppose $\text{int.cone}(P \cap \mathbb{Z}^{3t+1}) \cap Q$ is not empty. Let $y = y^{(1)} + \dots + y^{(n)}$ be a vector in this intersection with $y^{(i)} = (x^{(i)} \quad s^{(i)} \quad g^{(i)} \quad v^{(i)})^T \in P \cap \mathbb{Z}^{3t+1}$. Set x^* to be the concatenation of the $x^{(i)}$ -parts, i.e., $x^* = (x^{(1)}, \dots, x^{(n)})^T$. We claim that x^* is a solution of the uniform n -fold ILP that has objective value $\sum_{i=1}^n c^T(x^*)^{(i)} \geq T$. Due to the last constraint in P that sets the second to last coordinate of $y^{(i)}$ to $c^T y^{(i)}$ and the constraint in Q , we have $\sum_{i=1}^n c^T(x^*)^{(i)} = \sum_{i=1}^n c^T y^{(i)} \geq T$. The third group of constraints in P sets the g -part of the vectors to $Cy^{(i)}$ and since $y \in Q$, we have $\sum_{i=1}^n Cy^{(i)} = b^{(0)}$, so the global constraints $(C \ \dots \ C)x^* = b^{(0)}$ are fulfilled. The local constraints $B(x^*)^{(i)} = b^{(1)}$ are also fulfilled because of the first group of constraints in P and $y^{(i)} \in P$. Similarly, we have $0 \leq (x^*)^{(i)} \leq u$ for each $i \in [n]$, thanks to the second group of constraints in P . Clearly, x^* is integer. So x^* is a solution of the given uniform n -fold ILP with good enough objective value. \blacktriangleleft

We can now use this to obtain an algorithm for uniform n -fold ILPs:

► **Theorem 36.** *Uniform n -fold ILPs can be solved in time*

$$(\log(\max\{\|C\|_\infty, \|B\|_\infty, \|c\|_\infty\}))^{2^{O(r+s+t)}} \langle I \rangle^{O(1)}.$$

Proof. The parameters in our PQ-representation in Lemma 35 are: $\langle P \rangle, \langle Q \rangle, \log(m) = \langle I \rangle^{O(1)}$, $\|A^{(P)}\|_\infty = \max\{\|C\|_\infty, \|B\|_\infty, \|c\|_\infty\}$ and $K = \max\{M^{(P)}, M^{(Q)}, N\} = O(r+s+t)$. Then Corollary 24 yields running time:

$$\begin{aligned} & (\log(\|A^{(P)}\|_\infty))^{2^{O(K)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)} \log(m)^{O(1)} \\ &= (\log(\max\{\|C\|_\infty, \|B\|_\infty, \|c\|_\infty\}))^{2^{O(r+s+t)}} \langle I \rangle^{O(1)} \end{aligned}$$

\blacktriangleleft

A.5 MinSum-Weighted-BinPacking

In the MINSUM-WEIGHTED-BINPACKING problem, similar to scheduling, we are given an item vector $n \in \mathbb{N}^d$, a corresponding size vector $p \in \mathbb{N}_{>0}^d$, a weight vector $w \in \mathbb{N}_{>0}^d$ and a bin capacity $B \in \mathbb{N}_{>0}$. The goal is to pack all items into an unlimited number of bins such that the cost of all used bins is minimized; a bin with index i costs i times the total weight of the items in the bin. This problem has e.g. been studied in [15].

Note that it makes no sense to pack the $\|n\|_1$ items into more than $\|n\|_1$ bins, so we can via binary search (invoking an additional cost of $\log(\|n\|_1)$) assume that we know how many bins are used. Like in the other problems, we are looking at the decision variant, so we are also given a threshold for the objective value. Let T be this objective value and let m be the number of bins used by some feasible packing.

► **Lemma 37.** *MINSUM-WEIGHTED-BINPACKING has a PQ-representation.*

Proof. Let

$$P = \left\{ \begin{pmatrix} c \\ x \\ y \\ v \\ z \end{pmatrix} \in \mathbb{R}_{\geq 0}^{d+2m+2} \mid A \begin{pmatrix} c \\ x \\ y \\ v \\ z \end{pmatrix} \leq b \right\}$$

and

$$Q = \left\{ (n_1 \ \dots \ n_d \ 1 \ \dots \ 1 \ y \ v \ \tilde{T})^T \mid 0 \leq \tilde{T} \leq T \right\},$$

where A and b are defined by the following constraints:

$$\begin{aligned} p^T c &\leq B \\ \sum_{i=1}^m x_i &= 1 \\ \sum_{i=1}^d w_i c_i &= v \\ y_i + w_{\max} \min\{B, \|n\|_1\} &\geq v + x_i w_{\max} \min\{B, \|n\|_1\} && \forall i \in [m] \\ y_i &\leq w_{\max} \min\{B, \|n\|_1\} && \forall i \in [m] \\ \sum_{i=1}^m i y_i &= z \end{aligned}$$

The integer vectors in P fulfill the load constraint, i.e., the items (given by the vector c) fit into the bin. The second constraint assures that if x_i is non-negative and integer, exactly one component is equal to 1 and the others are zero. This selects the bin in which the items are to be packed. The third constraint just sets the variable v to the weight of the items in the bin/configuration. The fourth constraint assures that if $x_i = 1$, then $y_i \geq v$; otherwise, the constraint is always fulfilled, as $v \leq w_{\max} \min\{B, \|n\|_1\}$. The sixth constraint stores the actual value of the configuration (depending on the bin index and the weight of the items in the bin), while the fifth constraint just assures the boundedness of P (note that the other variables are also all bounded).

A solution of a given MINSUM-WEIGHTED-BINPACKING instance using m bins can be modelled by creating m vectors $y^{(1)}, \dots, y^{(m)} \in P \cap \mathbb{Z}^{d+2m+2}$, each of which represents a packing of a singular bin of the form $(c \ x \ y \ v \ z)^T$. Vector $y^{(k)}$ is defined as follows:

- The c -part represents the packed items,
- $x_k = 1$ and x_i for $i \neq k$ is set to 0,
- $y_k = v$ is set to the total weight of the items in the configuration, i.e., $\sum_{i=1}^d w_i c_i$ and the other y_i are set to 0 and
- z is set to kv , the weight of the configuration multiplied by its bin index.

Then each item is contained in one of the configurations, so the c -parts sum up to $(n_1, \dots, n_d)^T$; there is exactly one $y^{(i)}$ -vector for each bin, meaning the x -vectors sum up to $\mathbf{1}$; and finally, the objective values of all the configurations sum up to some value $\tilde{T} \leq T$. So $y := y^{(1)} + \dots + y^{(m)} \in \text{int.cone}(P \cap \mathbb{Z}^{d+2m+2}) \cap Q$.

For the other direction, consider a vector $y \in \text{int.cone}(P \cap \mathbb{Z}^{d+2m+2}) \cap Q$. Then due to the second constraint in P and the $\mathbf{1}$ in the description of Q , $y := y^{(1)} + \dots + y^{(m)}$ for some vectors $y^{(i)} \in P \cap \mathbb{Z}^{d+2m+2}$. By the above observations, each of these $y^{(i)}$ corresponds to a valid packing of a bin such that the z -entry contains an upper bound for the value this bin contributes to the overall objective. Since the sum of the c -vectors is equal to $(n_1, \dots, n_d)^T$, all items are packed and since the z -values sum up to a value $\tilde{T} \leq T$, the packing has a good enough objective value. ◀

► **Theorem 38.** *The MINSUM-WEIGHTED-BINPACKING problem can be solved in time*

$$(\log(\max\{p_{\max}, w_{\max} \min\{B, \|n\|_1\}\}))^{2^{O(d+m)}} \langle I \rangle^{O(1)}.$$

Proof. The parameters in our PQ-representation in Lemma 37 are: $\langle P \rangle, \langle Q \rangle, \log(m) = \langle I \rangle^{O(1)}$, $\|A^{(P)}\|_\infty = \max\{p_{\max}, w_{\max} \min\{B, \|n\|_1\}\}$ and $K = \max\{M^{(P)}, M^{(Q)}, N\} = O(d+m)$. Then Corollary 24 yields running time:

$$\begin{aligned} & (\log(\|A^{(P)}\|_\infty))^{2^{O(K)}} \langle P \rangle^{O(1)} \langle Q \rangle^{O(1)} \log(m)^{O(1)} \\ &= (\log(\max\{p_{\max}, w_{\max} \min\{B, \|n\|_1\}\}))^{2^{O(d+m)}} \langle I \rangle^{O(1)} \end{aligned}$$

◀

B Additive Approximation Scheme

Approximation algorithms have an error measure ε as part of their input and they usually produce a solution with value at most $(1 + \varepsilon)\text{OPT}$, where OPT is the optimum (for a minimization problem). An additive approximation algorithm on the other hand produces a solution with value at most $\text{OPT} + \varepsilon k$, where k is some parameter. A classical approximation algorithm can be seen as an additive approximation algorithm with additive error εOPT . So if we had two approximation algorithms with the same running time but one with additive error and one with multiplicative error, the one with additive error would be better, as long as $k < \text{OPT}$. Here, we consider parameter $k = p_{\max}$ for the problems $P|\{C_{\max}, C_{\min}, C_{\text{envy}}\}$.

In the additive approximation schemes presented in [4], Buchem et al. address load balancing problems on identical machines by introducing slot-MILPs, where jobs are organized in job size classes and fractionally assigned to machines such that the fractional components sum up to integral numbers of slots within each class. The idea is to first determine the number of slots to reserve for each class on each machine and to find an integral assignment to these slots in a second step. On each machine, the load is constrained to the target load interval $[\ell, u]$ for $\ell \in \mathbb{N}_0$ and $u \in \mathbb{N}_{>0}$. The problem of finding an assignment for $\|n\|_1$ jobs to m machines under the given load constraints is referred to as the *target load balancing problem*. The decision versions of $P|\{C_{\max}, C_{\min}, C_{\text{envy}}\}$ are special cases of this problem. An approximation scheme for each of these is included in [4] and also briefly described in Appendix B.2. In this section, we present an adaptation of the algorithm to a high-multiplicity setting. We slightly improve the running time from $m \|n\|_1^{O(\frac{1}{\varepsilon})}$ to $(mp_{\max})^{O(\frac{1}{\varepsilon})}$ and present a way to speed up the entire algorithm even more in case better running times can be achieved for the dynamic program making up the first part. First, we provide some more details on the original algorithm.

For $k \in [\frac{1}{\varepsilon}]$, $\frac{1}{\varepsilon} \in \mathbb{N}_{>0}$, a job size class \mathcal{J}_k refers to the set of jobs $\{j \in [\|n\|_1] \mid p_j \in ((k-1)\varepsilon p_{\max}, k\varepsilon p_{\max}]\}$. Formally, the slot-MILP is defined as follows.

$$\begin{aligned} \sum_{i \in [m]} x_{i,j} &= 1 & \forall j \in [\|n\|_1] \\ \ell &\leq \sum_{j \in [\|n\|_1]} p_j x_{i,j} \leq u & \forall i \in [m] \\ \sum_{i \in \mathcal{J}_k} x_{i,j} &= y_{i,k} & \forall i \in [m], k \in \left[\frac{1}{\varepsilon}\right] \\ x_{i,j} &\geq 0 & \forall j \in [\|n\|_1], i \in [m] \\ y_{i,k} &\in \mathbb{N}_0 & \forall i \in [m], k \in \left[\frac{1}{\varepsilon}\right] \end{aligned}$$

A weaker version, referred to as the slot-MILP $^\delta$, admits an error to the upper and lower bound respectively, using the redefined bounds $u' = u + \delta p_{\max}$ and $\ell' = \ell - \delta p_{\max}$ where $1 > \delta > 0$. While Buchem et al. [4] dedicate most of the paper to the δ -approximation, they also prove a structural property of the slot-MILP allowing to compute an exact solution in $m^{O(2^{\frac{1}{\varepsilon}})} \|n\|_1^{O(\frac{1}{\varepsilon} 2^{\frac{1}{\varepsilon}})}$. As our approach is based on the δ -approximation, we will almost exclusively refer to the slot-MILP $^\delta$ here.

Roughly outlined, the algorithm proceeds as follows to obtain a solution to the target load balancing problem:

1. For each $i \in [m]$ compute
 - a vector $y \in [\|n\|_1]^{\frac{1}{\varepsilon}}$ specifying how many jobs of each job size class are assigned to machine m_i and
 - a vector $z \in \mathbb{R}^{\frac{1}{\varepsilon}}$ specifying the average processing time of the jobs assigned to m_i within each job size class.

This step is achieved in $m \|n\|_1^{O(\frac{1}{\varepsilon})}$ time using a dynamic program which relies on some structural properties of an existing solution. If the slot-MILP $^\delta$ is feasible, the DP is guaranteed to find a solution, otherwise it states that the slot-MILP $^\delta$ is infeasible. As the vector z is only needed for the computation steps of the DP, it is discarded afterwards.

2. From the obtained vector y , assign jobs integrally to the slots using a local search algorithm described in Appendix B.1. This step has a running time of $O(\|n\|_1^5)$ and introduces an additional absolute error of at most εp_{\max} .

The overall running time is determined by the DP and amounts to $m \|n\|_1^{O(\frac{1}{\varepsilon})}$. In some cases, we can substantially decrease the number of jobs to be considered. Key to our approach is the following result by Govzmann et al. [19].

► **Lemma 14** (Govzmann et al. [19]). *For $P|\{C_{\max}, C_{\min}, C_{\text{envy}}\}$, there exists a kernel where the number of jobs of a specific type on a specific machine is bounded by $2p_{\max}$. So the load of every machine is bounded by $2p_{\max}^2 d$. The kernelization runs in $O(d)$ time.*

As described in more detail in Appendix C.3, this result follows from a couple of insights making it possible to preassign $\lceil \frac{np}{m} \rceil - p_{\max}$ jobs for each $p \in [d]$ to each machine and obtain a kernel consisting only of $O(mdp_{\max})$ jobs. Creating one single assignment vector to be shared among all machines is achieved in $O(d)$. Replacing $\|n\|_1$ by mdp_{\max} immediately yields the following:

► **Corollary 39.** *There exists an algorithm that, given an instance with d different processing times, computes a solution to the slot-MILP $^\delta$ in time $O((mdp_{\max})^{O(\frac{1}{\varepsilon})})$ or asserts that the slot-MILP $^\delta$ is infeasible.*

► **Remark 40.** The running time specified in [4] is $m^2 \left(\frac{\|n\|_1}{\delta \varepsilon} \right)^{O(\frac{1}{\varepsilon})}$. The full version of the paper [3] includes K different target load intervals $[\ell_i, u_i]$ for the machines. In other words, K is defined by $|\{[\ell_i, u_i] \mid i \in [m]\}|$. For each machine, the DP includes one additional step which consists of guessing the number of machines for each target load interval. This results in a running time of $m^K m \left(\frac{\|n\|_1}{\delta \varepsilon} \right)^{O(\frac{1}{\varepsilon})} = m^{K+1} \left(\frac{\|n\|_1}{\delta \varepsilon} \right)^{O(\frac{1}{\varepsilon})}$. It appears that K was replaced by 1 in the simplified version with only one target load interval, leading to the squared number of machines. However, this would imply guessing the number of machines for the remaining target load interval. Since the step of guessing the number of machines can be entirely omitted, it seems that the running time should be $m \left(\frac{\|n\|_1}{\delta \varepsilon} \right)^{O(\frac{1}{\varepsilon})}$ instead.

As the running time of the DP hides the running time of the local search algorithm used to find an integral assignment to the slots, using Corollary 39 directly yields (note that $d \leq p_{\max}$):

► **Theorem 3.** *There is an additive approximation scheme for $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$ with error at most εp_{\max} that has running time $(mp_{\max})^{O(\frac{1}{\varepsilon})}$.*

However, it might be possible to adapt the DP even further such that the number of machines disappears from the exponential term. In the hope that further research will reveal ways of speeding up the DP enough for the running time of the local search to become relevant, we present an enhanced local search algorithm.

B.1 Local search algorithm

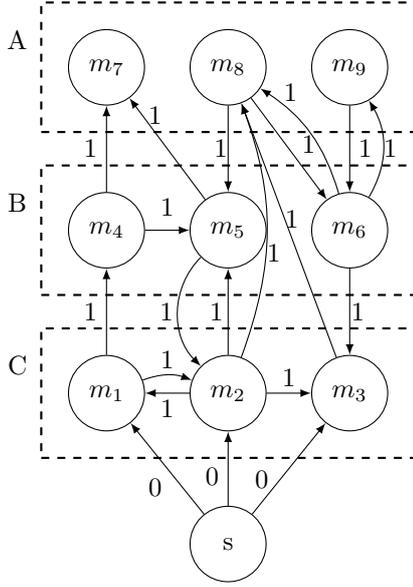
Given an exact solution or a δ -approximation to the slot-MILP, Buchem et al. [4] use a local search algorithm to compute an integral solution with error at most εp_{\max} to the upper and lower bounds respectively. More precisely, from an exact solution to the slot-MILP, the algorithm computes an integral schedule such that $\sum_{j \in [n]_{\|1}} p_j x_{i,j} \in [\ell - \varepsilon p_{\max}, u + \varepsilon p_{\max}]$

holds for each machine $i \in [m]$.

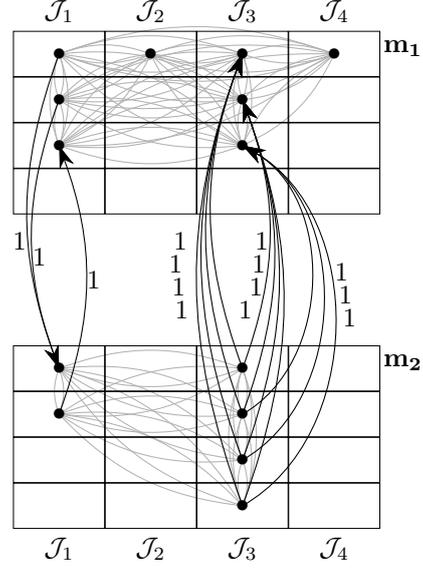
Starting from a δ -approximation, the local search adds an error of εp_{\max} to the error already existing due to the DP. Each machine has therefore a load in the interval $[\ell - \delta p_{\max} - \varepsilon p_{\max}, u + \delta p_{\max} + \varepsilon p_{\max}]$ on termination of the local search. The error contributed by the δ -approximation can be compensated by setting $\delta = \frac{\varepsilon^*}{2}, \varepsilon = \frac{\varepsilon^*}{2}$ for some final error parameter $\varepsilon^* > 0$.

Based on the vector y , the algorithm arbitrarily assigns $y_{i,k}$ jobs of class \mathcal{J}_k to machine i for each $i \in [m], k \in [\frac{1}{\varepsilon}]$. This does not guarantee a load within the interval $[\ell - \varepsilon p_{\max}, u + \varepsilon p_{\max}]$ on any machine though. Therefore, the algorithm proceeds by exchanging jobs on overloaded machines for smaller jobs of the same class on machines with load at most u . Machines with a load between $u + 1$ and $u + \varepsilon p_{\max}$ cannot provide any smaller jobs for an exchange because they might become overloaded themselves. When such machines are encountered during the search, they are repaired in a recursive procedure. Dealing with the overloaded machines is referred to as the first stage. In a second stage, an analogous procedure is performed for the machines with a load smaller than $\ell - \varepsilon p_{\max}$. Since both stages follow the same principles, we will only discuss the first stage.

Buchem et al. [4] provide a representation of the local search algorithm as a weighted directed graph $G = (V, E)$. For each slot on each machine, a vertex is created. A machine is described by a clique where each pair of vertices is connected by a bidirectional edge of weight 0. For each pair of slots $v, w \in V$, the graph contains an edge from v to w of weight 1 if and only if both slots are in the same job size class and on different machines and w contains a job smaller than the one assigned to v , that is, if slot w contains a possible swap candidate for v . Furthermore, V contains an additional source vertex s with an edge of weight 0 to all vertices on all machines with a load exceeding $u + \varepsilon p_{\max}$. For an illustration, see Figure 1a and Figure 1b. The local search algorithm is equivalent to a breadth-first search (BFS) starting at s where the vertices representing one machine are collected all at once as soon as one vertex of the clique is reached. A swap of jobs between two slots is expressed by redirecting all incoming and outgoing edges accordingly. Whenever two jobs are exchanged, the search restarts at vertex s . Denote by $\delta(v)$ the number of edges of weight 1 on a path from s to vertex v and let $j_1, \dots, j_{\|n\|_1}$ be the jobs in increasing order of job size. In [4], it is shown that the value of the following potential function increases with each swap:



(a) A graph representation of the local search algorithm. A: Machines with load at most u . B: Machines with a load in the interval $[u + 1, u + \varepsilon p_{\max}]$. C: Machines with a load exceeding $u + \varepsilon p_{\max}$. In this example, m_1 cannot exchange any jobs before m_4 or m_2 is repaired.



(b) A close up view showing the connections between two machines m_1 and m_2 . Bullet points: Jobs assigned to slots. Gray lines: Edges of weight 0 connecting jobs on one machine. In this example, the job of class J_3 in row 3 on m_1 has the same size as the one of the same class in row 1 on m_2 . Therefore, there is no edge between the corresponding bullet points.

$$\sum_{i=1}^{\|n\|_1} i\delta(j_i)$$

From this function, an upper bound of $\|n\|_1^2 m$ can be derived ($\|n\|_1^3$ in the original paper). A BFS can be carried out in $O(\|n\|_1^2)$. The preprocessing reduces the number of jobs considered in the local search algorithm. This improves the running time $O(\|n\|_1^4 m)$ to $O((mdp_{\max})^4 m)$. However, we can do even better by exploiting some structural properties of the graph given in this specific setting.

► **Lemma 41.** $\delta(v) < d$ for each $v \in V$.

Proof. For the proof, we have to distinguish between vertices adjacent to s and the remaining ones. If a vertex $v \in V$ corresponds to a slot on a machine with a load exceeding $u + \varepsilon p_{\max}$, then $\delta(v) = 0$ regardless of the processing time of the job stored in the slot. Let $v_1, v_2 \in V$ and $(s, v_1), (s, v_2) \notin E$ and denote by $p(v')$ the processing time of the job assigned to the slot represented by a vertex $v' \in V$. Let $w(e)$ be the weight of an edge $e \in E$. Then from $p(v_1) = p(v_2)$ follows the equality of the sets $\{x \in V \mid (x, v_1) \in E \wedge w((x, v_1)) = 1\}$ and $\{x \in V \mid (x, v_2) \in E \wedge w((x, v_2)) = 1\}$, implying that vertices with distance > 0 storing identical processing times belong to the same BFS layer. At the same time, all vertices inside a machine clique also belong to the same BFS layer, implying that a path does not contain any detours inside cliques. Therefore, each processing time can appear at most once along a path. From this, we can conclude that $\delta(v)$ is at most $d - 1$ for each $v \in V$. ◀

Based on this insight, we can make the following adjustment to the potential function.

$$\sum_{i=1}^d i \sum_{j=1}^{n_i} \delta(j_i) \quad (1)$$

where $1, \dots, d$ represents the sequence of indices of job sizes arranged in ascending order.

Because at most $2p_{\max}$ jobs of one specific size are assigned to each machine, we have $n_i \leq 2mp_{\max}$ for each $1 \leq i \leq d$. The first sum contributes a factor d^2 and $\delta(j_i)$ is bounded by d due to Lemma 41. From this, we can bound the potential function by $d^3 2mp_{\max}$.

► **Lemma 42.** *The value of (1) increases with each swap.*

Proof. Let $j_{v_1}, j_{v_2} \in [n]_1$ be two jobs to be swapped and let $v_1, v_2 \in V$ be the corresponding vertices. Let $p_{j_{v_1}}, p_{j_{v_2}}$ be the processing times of jobs j_{v_1} and j_{v_2} and let $p_{j_{v_1}} > p_{j_{v_2}}$. As in [4], we will start by showing that a swap does not decrease the distance to any vertex. Note that no edges outgoing from s are added, since a swap is only carried out with machines with load at most u and the difference in processing time between two jobs of one size class is at most $\varepsilon p_{\max} - 1$. Exchanging the jobs between v_1 and v_2 causes a reversal of the edge (v_1, v_2) and a complete exchange of both the sets of outgoing edges of weight 1 and the sets of incoming edges of weight 1 between v_1 and v_2 .

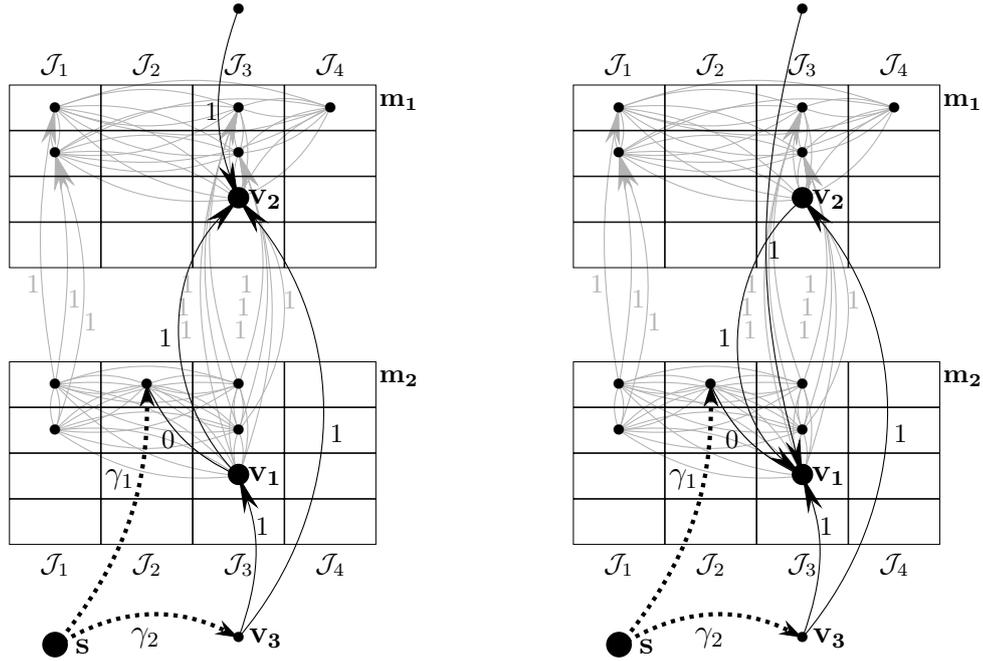
Let $\delta(u), \delta'(u)$ denote the distance from s to some vertex $u \in V$ before and after the swap, respectively. It holds that $\delta(v_2) = \delta(v_1) + 1$, otherwise both v_1 and v_2 would belong to one and the same BFS layer, preventing the swap between v_1 and v_2 in the first place. Another important observation is that the edges within the cliques do not change. A new path to v_1 can therefore only be provided by an edge previously incident to v_2 and vice versa. Suppose that the distance to v_1 decreases. Then there must have been a path leading to v_2 such that $\delta(v_2) \leq \delta(v_1) - 1$, a contradiction to $\delta(v_2) = \delta(v_1) + 1$. Suppose that the distance to v_2 decreases. Then one of the edges originally incident to v_1 yields a new path to v_2 such that $\delta'(v_2) \leq \delta(v_2) - 1 = \delta(v_1)$. Let (v_3, v_1) be this edge before the swap. Because of the transitive relation between all the edges of weight 1, there must have been an edge from v_3 to v_2 too, implying $\delta(v_2) = \delta(v_1)$. This scenario is illustrated in Figure 2a and Figure 2b.

All paths in the graph which include neither v_1 nor v_2 are unaffected by the swap. Those including at least one of the vertices v_1 or v_2 cannot become shorter by the swap since neither $\delta(v_1)$ nor $\delta(v_2)$ decreased.

We will now inspect the changes caused to the value of the potential function by the swap. Let $\pi : [p_{\max}] \rightarrow [d]$ be a function mapping a processing time to its position in an ascending sequence of all processing times in $[d]$. As established before, $\delta'(u) \geq \delta(u)$ for all $u \in V$. Therefore, we have the following inequality for the potential without the swapped jobs.

$$\sum_{i=1}^d i \sum_{j=1}^{n_i} \delta(j_i) - \pi(j_{v_1})\delta(j_{v_1}) - \pi(j_{v_2})\delta(j_{v_2}) \leq \sum_{i=1}^d i \sum_{j=1}^{n_i} \delta'(j_i) - \pi(j_{v_1})\delta'(j_{v_1}) - \pi(j_{v_2})\delta'(j_{v_2})$$

For the exchanged jobs, we can derive the following inequality from $\delta(j_{v_2}) = \delta(j_{v_1}) + 1$.



(a) Two machines m_1 and m_2 before the swap. The edge $(\mathbf{v}_3, \mathbf{v}_2)$ is implied by $(\mathbf{v}_3, \mathbf{v}_1)$ and $(\mathbf{v}_1, \mathbf{v}_2)$ because all edges of weight 1 are based on the transitive relation $>$. γ_1 : shortest path to \mathbf{v}_1 . γ_2 : shortest path to \mathbf{v}_3 . For the swap to be carried out, γ_1 must be shorter than $|\gamma_2| + 1$.

(b) The machines after the swap. All the incoming edges of weight 1 have been exchanged between \mathbf{v}_1 and \mathbf{v}_2 and the edge $(\mathbf{v}_1, \mathbf{v}_2)$ has been flipped.

$$\pi(j_{v_1})\delta'(j_{v_1}) + \pi(j_{v_2})\delta'(j_{v_2}) = \pi(j_{v_1})\delta(j_{v_2}) + \pi(j_{v_2})\delta(j_{v_1}) \quad (2)$$

$$= (\pi(j_{v_1}) - \pi(j_{v_2}))\delta(j_{v_2}) + \pi(j_{v_2})\delta(j_{v_2}) + \pi(j_{v_2})\delta(j_{v_1}) \quad (3)$$

$$> (\pi(j_{v_1}) - \pi(j_{v_2}))\delta(j_{v_1}) + \pi(j_{v_2})\delta(j_{v_2}) + \pi(j_{v_2})\delta(j_{v_1}) \quad (4)$$

$$= \pi(j_{v_1})\delta(j_{v_1}) - \pi(j_{v_2})\delta(j_{v_1}) + \pi(j_{v_2})\delta(j_{v_2}) + \pi(j_{v_2})\delta(j_{v_1}) \quad (5)$$

$$= \pi(j_{v_1})\delta(j_{v_1}) + \pi(j_{v_2})\delta(j_{v_2}) \quad (6)$$

Hence, the value of the potential function increases by at least 1 with each swap, implying that the number of swaps is bounded by $d^3 2mp_{\max}$. \blacktriangleleft

► **Lemma 43.** *One BFS run and the subsequent swap can be completed in $O(\frac{1}{\varepsilon}m^2 + \log(dp_{\max}))$ time.*

Proof. The running time of a BFS is given by $O(|V| + |E|)$. The naive approach consists of including all edges of the graph in the search. As $|E|$ is only bounded by $|V|^2$, this yields running time $O((mdp_{\max})^2)$ for one BFS run. However, we can speed up this procedure based on the following thoughts.

1. The exact slots or size classes along one path are irrelevant before a machine with load at most u is reached.
2. Edges of weight 1 exist only between slots of the same class.

3. Let $\mathcal{P}_{i,k}$ be the partition assembling the vertices representing the $y_{i,k}$ slots of class \mathcal{J}_k for some machine $i \in [m]$. The edges of weight 1 are based on the strict order $>$. Hence, it is sufficient to consider the outgoing edges of the vertex storing the biggest job of some class \mathcal{J}_k assigned to some machine $i \in [m]$ in order to know all the destination machines of the outgoing edges from partition $\mathcal{P}_{i,k}$.
4. For the same reason, the incoming edges of the vertex storing the smallest job of some class \mathcal{J}_k assigned to some machine $i \in [m]$ reveal all the machines from which partition $\mathcal{P}_{i,k}$ can be reached by at least one edge.

From observation 1 we can conclude that it would be convenient to carry out the search on a graph similar to the one displayed in Figure 1a, that is, a graph where slots and size classes are abstracted away.

In order to construct and maintain such a graph, we first create an auxiliary data structure consisting of m arrays of size $\frac{1}{\varepsilon}$. Each array represents a machine and each cell represents a size class \mathcal{J}_k for $k \in [\frac{1}{\varepsilon}]$ and contains a min-max heap $T_{i,k}$ storing the jobs assigned to the $y_{i,k}$ slots on the corresponding machine $i \in [m]$. Denote by $\min(T_{i',k'})$ ($\max(T_{i',k'})$) the smallest (resp. biggest) element of some heap $T_{i',k'}$. Let $i_1, i_2 \in [m]$, $k_1, k_2 \in [\frac{1}{\varepsilon}]$. Denote by C_i the clique describing machine $i \in [m]$ in the original graph G . Then G contains an edge leading from some vertex in C_{i_1} to some vertex in C_{i_2} if and only if $\max(T_{i_1,k_1}) > \min(T_{i_2,k_2})$ and $k_1 = k_2$. Creating the auxiliary data structure requires the insertion of $O(mdp_{\max})$ jobs into min-max heaps, which can be done in $O(mdp_{\max} \log(mdp_{\max}))$ time.

We now have all the ingredients needed to construct our simplified graph $G' = (V', E')$ consisting of m vertices, each one representing some machine $i \in [m]$. Before each BFS run, we update G' by iterating through the job size classes, comparing machines pairwise within each class. If the largest job on one machine is larger than the smallest job on some other machine, the edge in G' is added accordingly.

The swap procedure as a whole now consists of the following steps:

- Update G' in $O(\frac{1}{\varepsilon}m^2)$.
- Perform the BFS in $O(m^2)$.
- When some machine i_1 suitable for a swap is found, that is, when its load is at most u , then perform a swap between i_1 and its predecessor i_2 in the BFS. For this purpose, iterate through the $\frac{1}{\varepsilon}$ classes. For each $k \in [\frac{1}{\varepsilon}]$ compare the biggest job of i_2 with the smallest job on i_1 using the auxiliary data structure described above. If $\max(T_{i_2,k}) > \min(T_{i_1,k})$, then perform the swap of the two just compared elements. Iterating through the classes contributes an additive term of $\frac{1}{\varepsilon}$ to the running time. Swapping jobs involves two remove and insert operations, each of them running in $O(\log(dp_{\max}))$. Therefore, this step adds $O(\frac{1}{\varepsilon} + \log(dp_{\max}))$ to the running time.

Assembling all the terms yields running time $O(\frac{1}{\varepsilon}m^2) + O(m^2) + O(\frac{1}{\varepsilon} + \log(dp_{\max})) = O(\frac{1}{\varepsilon}m^2 + \log(dp_{\max}))$ for one BFS run and the subsequent swap. ◀

► **Proposition 44.** *Given a solution to the slot-MILP, an integral solution to the target load balancing problem with an absolute error of at most εp_{\max} can be computed in time $O(m^3 d^3 p_{\max} \frac{1}{\varepsilon} \log(mdp_{\max}))$.*

Proof. Due to Lemma 42, the number of swaps is bounded by $d^3 2mp_{\max}$. By Lemma 43, one swap and the preceding BFS run can be carried out in $O(\frac{1}{\varepsilon}m^2 + \log(dp_{\max}))$. Constructing the auxiliary data structure needed for the graph can be achieved in $O(mdp_{\max} \log(mdp_{\max}))$. Putting this together, we obtain $O(mdp_{\max} \log(mdp_{\max})) + d^3 2mp_{\max} O(\frac{1}{\varepsilon}m^2 + \log(dp_{\max})) = O(m^3 d^3 p_{\max} \frac{1}{\varepsilon} \log(mdp_{\max}))$. ◀

B.2 Application

The idea by Buchem et al. [4] is to guess the target load intervals for either $P||C_{\max}$, $P||C_{\min}$ or $P||C_{\text{envy}}$ in steps of εp_{\max} . The local search is then performed for the best solution obtained by the DP. Based on our insights, we can make the following adjustments to the running times presented in [4].

For $P||C_{\max}$, ℓ is set to 0 and u is guessed within the interval $[\frac{1}{m} \sum_{j=1}^{\|n\|_1} p_j, \frac{1}{m} \sum_{j=1}^{\|n\|_1} p_j + p_{\max}]$.

► **Corollary 45.** *There exists an algorithm for $P||C_{\max}$ with running time $(mp_{\max})^{O(\frac{1}{\varepsilon})}$ computing a schedule with makespan at most $\text{OPT} + \varepsilon p_{\max}$.*

For $P||C_{\min}$, u is set to $\frac{1}{m} \sum_{j=1}^{\|n\|_1} p_j$ and ℓ is guessed within the interval $[\frac{1}{m} \sum_{j=1}^{\|n\|_1} p_j - p_{\max}, \frac{1}{m} \sum_{j=1}^{\|n\|_1} p_j]$.

► **Corollary 46.** *There exists an algorithm for $P||C_{\min}$ with running time $(mp_{\max})^{O(\frac{1}{\varepsilon})}$ computing a schedule where each machine has load at least $\text{OPT} - \varepsilon p_{\max}$.*

For $P||C_{\text{envy}}$, both ℓ and u are guessed from the intervals above.

► **Corollary 47.** *There exists an algorithm for $P||C_{\text{envy}}$ with running time $(mp_{\max})^{O(\frac{1}{\varepsilon})}$ computing a schedule with envy value at most $\text{OPT} + \varepsilon p_{\max}$.*

C Omitted proofs

Here, we give some proofs that were omitted due to space constraints.

C.1 Miscellaneous

This part covers several smaller observations like solving optimization problems via decision problems.

Frank Tardos

► **Corollary 13** (Stated in a similar form by Etscheidt et al. [16]). *For every $w \in \mathbb{R}^N$, $b \in \mathbb{R}$, $\Delta \in \mathbb{N}$, one can compute \bar{w} , \bar{b} with $\|\bar{w}\|_{\infty}, |\bar{b}| \leq (N\Delta)^{O(N^3)}$ in time $N^{O(1)}$ such that for every $x \in [-\Delta, \Delta]^N$, $w^T x \leq b \iff \bar{w}^T x \leq \bar{b}$.*

Proof. Apply Theorem 12 to the vector $\begin{pmatrix} w \\ b \end{pmatrix}$ and the bound $N\Delta + 1$ to obtain a vector $\begin{pmatrix} \bar{w} \\ \bar{b} \end{pmatrix}$.

Now, let $x \in [-\Delta, \Delta]^N$ and consider the vector $x' = \begin{pmatrix} x \\ -1 \end{pmatrix}$. Then $\|x'\|_1 = \|x\|_1 + 1 \leq N\Delta + 1$,

so we have $\text{sign}\left(\begin{pmatrix} w \\ b \end{pmatrix}^T x'\right) = \text{sign}\left(\begin{pmatrix} \bar{w} \\ \bar{b} \end{pmatrix}^T x'\right)$ by Theorem 12. Now:

$$\begin{aligned} w^T x \leq b &\iff w^T x - b \leq 0 \iff \begin{pmatrix} w \\ b \end{pmatrix}^T \begin{pmatrix} x \\ -1 \end{pmatrix} \leq 0 \\ &\iff \begin{pmatrix} w \\ b \end{pmatrix}^T x' \leq 0 \end{aligned}$$

This is equivalent to:

$$\begin{aligned}
\text{sign}\left(\begin{pmatrix} w \\ b \end{pmatrix}^T x'\right) \in \{0, +1\} &\iff \text{sign}\left(\begin{pmatrix} \bar{w} \\ \bar{b} \end{pmatrix}^T x'\right) \in \{0, +1\} \\
&\iff \begin{pmatrix} \bar{w} \\ \bar{b} \end{pmatrix}^T x' \leq 0 \\
&\iff \begin{pmatrix} \bar{w} \\ \bar{b} \end{pmatrix}^T \begin{pmatrix} x \\ -1 \end{pmatrix} \leq 0 \\
&\iff \bar{w}^T x - \bar{b} \leq 0 \\
&\iff \bar{w}^T x \leq \bar{b}
\end{aligned}$$

The running time is the same as in Theorem 12, namely $(N+1)^{O(1)} = N^{O(1)}$. We have $\|\bar{w}\|_\infty, |\bar{b}| \leq (N\Delta+1)^{O((N+1)^3)} = (N\Delta)^{O(N^3)}$. \blacktriangleleft

The Exercise From [11] (Hint 3.18)

► **Lemma 48.** *For computable f and any $\beta, \gamma \in \mathbb{N}$, we have $\log(\beta)^{f(\gamma)} \leq 2^{\frac{(f(\gamma))^2}{2}} \beta^{o(1)}$ for some computable g .*

Proof. We have

$$\log(\beta)^{f(\gamma)} = 2^{\log(\log(\beta))f(\gamma)} \leq 2^{\frac{(\log(\log(\beta)))^2 + (f(\gamma))^2}{2}} = 2^{\frac{(\log(\log(\beta)))^2}{2}} 2^{\frac{(f(\gamma))^2}{2}} = 2^{\frac{(f(\gamma))^2}{2}} \beta^{o(1)}.$$

The second transformation uses that $ab \leq \frac{a^2+b^2}{2}$ (which is equivalent to $(a-b)^2 \geq 0$) and the last transformation uses $\log(\log(\beta))^2 = o(\log(\beta))$. \blacktriangleleft

Optimization vs. Decision Problems

The optimization problems considered in this paper can be easily solved by solving several decision problems, simply by doing a binary search over some interval $[0, v]$. The key ingredient is that the objective value is always bounded by some value v that depends on the numbers in the input and hence $\log(v) \leq \langle I \rangle^{O(1)}$. For uniform n -fold ILPs, MINSUM-WEIGHTED-BINPACKING and $\sum w_j U_j$, we just have to find a solution that has at least (or at most) some value T that is given by the binary search. For C_{\max} and C_{\min} , we then have an upper bound u or a lower bound ℓ for the load of the machines and have to decide whether a schedule exists with load values in $[0, u]$ or $[\ell, v]$. For C_{envy} , it is a little more tricky than for C_{\max} and C_{\min} , because a ‘guess’ T for the envy-value only gives us the difference $u - \ell$, not the actual values ℓ and u we will need for the framework. The following lemma solves this predicament:

► **Lemma 49.** *For $P \in \{C_{\max}, C_{\min}, C_{\text{envy}}\}$, there always exists an optimal solution in which each machine i has load $L_i \in [L - p_{\max}, L + p_{\max}]$, where L is the sum of all processing times divided by m .*

Proof. Suppose there is an optimal schedule σ that does not fulfill this property. Then σ can be improved by moving any job from a machine j with the highest load to a machine k with the lowest load. This increases the load of k by at most p_{\max} and reduces the load of j by at most p_{\max} . Note that the load of k has to be $\leq L$ and the load of j has to be $\geq L$. So this modification of the schedule can only either move the load value of the two machines into the interval $[L - p_{\max}, L + p_{\max}]$ or move it around inside; a load value cannot

fall out of this interval. Note that this modification can only improve the objective value (w.r.t. either of the three objectives). So iterating this procedure will lead to a schedule that has the desired property and is still optimal. ◀

This implies that we can assume a given upper/lower bound u or l for the load value to lie in the interval $[L - p_{\max}, L + p_{\max}]$. Otherwise, we can decide the feasibility of the problem in polynomial time (with a greedy algorithm). In particular, for the C_{\min} -objective, we only have to consider configurations with a load value that is at most $\ell + 2p_{\max}$. Note that by the same argument, one can even show that the difference between the largest load and the smallest load can be at most p_{\max} (without the factor 2).

For objectives C_{\max} and C_{\min} , reducing optimization to decision adds a factor $\log(p_{\max}) \leq \langle I \rangle^{O(1)}$ to the running time. For C_{envy} , we guess the minimum load l , which can have at most $O(p_{\max})$ different values. We then set $u = \ell + T$ and solve the decision problem. Overall, this adds the same factor $\langle I \rangle^{O(1)}$ for the binary search, and a factor $O(p_{\max})$ to the running time, which yields a total factor of $\langle I \rangle^{O(1)} p_{\max}$. Note that this only works if there are no additional constraints like class or capacity constraints or setup times, as Lemma 49 no longer works directly. We might have to try out more values in those cases to solve the optimization problem.

C.2 Proximity

► **Theorem 23.** *Consider a problem with a PQ-representation (P, Q, m) , where $P = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(P)}x = b^{(P)}\}$ with $A^{(P)} \in \mathbb{Z}^{M^{(P)} \times N}$ and $Q = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(Q)}x = b^{(Q)}\}$ with $A^{(Q)} \in \mathbb{Z}^{M^{(Q)} \times N}$. Then by solving the convexified relaxation of the corresponding ILP in time $2^{O((M^{(Q)}+N)^2)} (M^{(P)} \log(\max\{\|A^{(P)}\|_{\infty}, \|A^{(Q)}\|_{\infty}, \|b^{(P)}\|_{\infty}, \|b^{(Q)}\|_{\infty}\}m))^{O(1)}$ and modifying P , we can assume that the points in P have ℓ_{∞} -norm at most*

$$((M^{(Q)} + N)M^{(P)} \max\{\|A^{(P)}\|_{\infty}, \|A^{(Q)}\|_{\infty}\})^{O((M^{(Q)}+N)M^{(P)})}.$$

Proof. First, consider the formulation of the ILP in Lemma 20 and its parameters. We have $n = O(m)$ as the number of blocks, $\tau = 2$ as the number of different blocks, $t = N$ as the number of variables per block, $s = M^{(P)}$ as the number of constraints per block, $r = M^{(Q)} + N$ as the number of global constraints and the largest entry in the matrix is $\Delta = \max\{\|A^{(P)}\|_{\infty}, \|A^{(Q)}\|_{\infty}\}$. It should be noted that there are upper bounds $x^{(i)} \leq u^{(i)}$ in the convexified relaxation, but the variables in the ILP do not have explicit upper bounds. However, as we know by the definition of PQ-representations, P and Q themselves are bounded. Hence, with Cramer's rule [8] we can bound the points in P and Q , i.e., the solutions of the ILP. In particular, the solutions are bounded by $\Delta^{N-1} \max\{\|b^{(P)}\|_1, \|b^{(Q)}\|_1\}$ if we use the Hadamard inequality [21] to bound the determinants. This yields (identical) artificial upper bounds $u^{(i)} = u$ with $\|u\|_{\infty} = \Delta^{N-1} \max\{\|b^{(P)}\|_1, \|b^{(Q)}\|_1\}$. Hence, the largest number in the formulation is $K = \max\{\|A^{(P)}\|_{\infty}, \|A^{(Q)}\|_{\infty}, \|b^{(P)}\|_{\infty}, \|b^{(Q)}\|_{\infty}, \|u\|_{\infty}\}$.

So using Proposition 21, we can solve the convexified relaxation in time:

$$\begin{aligned} & \tau t^{O(t)} 2^{O(\tau^2)} (sr \log(Kn))^{O(1)} \\ &= 2N^{O(N)} 2^{O((M^{(Q)}+N)^2)} \\ & (M^{(P)}(M^{(Q)} + N) \log(\max\{\|A^{(P)}\|_{\infty}, \|A^{(Q)}\|_{\infty}, \|b^{(P)}\|_{\infty}, \|b^{(Q)}\|_{\infty}, \|u\|_{\infty}\}m))^{O(1)} \\ &= 2^{O((M^{(Q)}+N)^2)} (M^{(P)} \log(\max\{\|A^{(P)}\|_{\infty}, \|A^{(Q)}\|_{\infty}, \|b^{(P)}\|_{\infty}, \|b^{(Q)}\|_{\infty}\}m))^{O(1)} \end{aligned}$$

Suppose that x^* is a solution of the convexified relaxation such that $(x^*)^{(1)} = \dots = (x^*)^{(m)}$. This can be obtained from any solution computed by the algorithm described in Proposition 21: Just replace the first m solution parts by their overall average. As they all lie in the convex hull of the same points (see the description of the algorithm before Proposition 21), the average also lies in the convex hull and is hence feasible.

By Theorem 22, there is a solution z^* to the ILP from Lemma 20 such that

$$\begin{aligned} \|x^* - z^*\|_1 &\leq (r\Delta(s\Delta)^{O(s)})^{O(r)} \\ &= (rs\Delta)^{O(rs)} \\ &\leq ((M^{(Q)} + N)M^{(P)} \max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\})^{O((M^{(Q)}+N)M^{(P)})} \end{aligned}$$

Let $D := ((M^{(Q)} + N)M^{(P)} \max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\})^{O((M^{(Q)}+N)M^{(P)})}$. Then we can assume that the variables in the ILP are bounded by $(x^*)^{(i)} - D \leq z^{(i)} \leq (x^*)^{(i)} + D$. So after modifying the right-hand-side $b^{(P)}$ to $b^{(P)} + (x^*)^{(i)}$, the variables are bounded by D in ℓ_∞ -norm. Note that by assumption, the solutions $(x^*)^{(i)}$ are the same for all blocks but the last one. So modifying the right-hand-side and re-translating the ILP to a PQ-representation (using Lemma 20) yields one modified polytope P . ◀

► **Theorem 5.** *If a problem has a PQ-representation (P, Q, m) with $P = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(P)}x = b^{(P)}\}$ and $Q = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(Q)}x = b^{(Q)}\}$ with $A^{(P)} \in \mathbb{Z}^{M^{(P)} \times N}$, $A^{(Q)} \in \mathbb{Z}^{M^{(Q)} \times N}$, $b^{(P)} \in \mathbb{Z}^{M^{(P)}}$ and $b^{(Q)} \in \mathbb{Z}^{M^{(Q)}}$, it can be solved in time*

$$((M^{(P)}M^{(Q)} \log(\max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\}))^{2^{O(N)}} + 2^{O((M^{(Q)})^2)})(\langle P \rangle \langle Q \rangle \log(m))^{O(1)}.$$

Proof. First, we use Theorem 23 to solve the convexified relaxation of the ILP corresponding to (P, Q, m) and modify $b^{(P)}$ in time

$$2^{O((M^{(Q)}+N)^2)}(M^{(P)} \log(\max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty, \|b^{(P)}\|_\infty, \|b^{(Q)}\|_\infty\}m))^{O(1)}$$

and restrict ourselves to points in P that have ℓ_∞ -norm at most

$$D := ((M^{(Q)} + N)M^{(P)} \max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\})^{O((M^{(Q)}+N)M^{(P)})}$$

by adding explicit bounds to the constraints in $A^{(P)}x = b^{(P)}$. Note that this increases the number of constraints to $M^{(P)} + 2N$ and the number of variables to $N + M^{(P)} + 2N = 3N + M^{(P)}$ due to adding slack variables. We then use the result by Frank and Tardos (see Corollary 13) to reduce the coefficients in time $(3N + M^{(P)})^{O(1)}$. Afterwards, the coefficients in $A^{(P)}$ and $b^{(P)}$ are bounded by $((3N + M^{(P)})D)^{O((3N+M^{(P)})^3)}$. Using Proposition 16, the given problem can be solved in time $\langle P' \rangle^{2^{O(N)}} \langle Q \rangle^{O(1)} \log(m)^{O(1)}$. Note that the modified polytope P' has encoding length at most

$$\begin{aligned} \langle P' \rangle &\leq O((3N + M^{(P)})(M^{(P)} + 2N) \log(\max\{\|A^{(P)}\|_\infty, \|b^{(P)}\|_\infty\})) \\ &= O((3N + M^{(P)})(M^{(P)} + 2N) \log(((3N + M^{(P)})D)^{O((3N+M^{(P)})^3)})) \\ &= O((3N + M^{(P)})^4(M^{(P)} + 2N) \log((3N + M^{(P)})D)) \end{aligned}$$

So the problem itself can then be solved in time:

$$\begin{aligned}
& \langle P' \rangle^{2^{O(N)}} \langle Q \rangle^{O(1)} \log(m)^{O(1)} \\
&= ((3N + M^{(P)})^4 (M^{(P)} + 2N) \log((3N + M^{(P)})D))^{2^{O(N)}} \langle Q \rangle^{O(1)} \log(m)^{O(1)} \\
&\leq (M^{(P)} \log(((M^{(Q)} + N)M^{(P)} \max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\})^{O((M^{(Q)}+N)M^{(P)})}))^{2^{O(N)}} \\
&\quad \langle Q \rangle^{O(1)} \log(m)^{O(1)} \\
&\leq (M^{(P)} M^{(Q)} \log(\max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\}))^{2^{O(N)}} (\langle Q \rangle \log(m))^{O(1)}
\end{aligned}$$

Altogether (with the preprocessing), this yields an algorithm running in time

$$\begin{aligned}
& (M^{(P)} M^{(Q)} \log(\max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\}))^{2^{O(N)}} (\langle Q \rangle \log(m))^{O(1)} \\
&+ 2^{O((M^{(Q)}+N)^2)} (M^{(P)} \log(\max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty, \|b^{(P)}\|_\infty, \|b^{(Q)}\|_\infty\} m))^{O(1)} \\
&+ N^{O(1)} \\
&= ((M^{(P)} M^{(Q)} \log(\max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\}))^{2^{O(N)}} + 2^{O((M^{(Q)}+N)^2)}) (\langle P \rangle \langle Q \rangle \log(m))^{O(1)}
\end{aligned}$$

◀

As a side note, Theorem 23 is essentially a preprocessing algorithm like the one by Govzmann et al. [19] and produces a kernel for certain problems that have a PQ-representation:

► **Corollary 50.** *Consider a problem with a PQ-representation (P, Q, m) , where $P = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(P)}x = b^{(P)}\}$ with $A^{(P)} \in \mathbb{Z}^{M^{(P)} \times N}$ and $Q = \{x \in \mathbb{R}_{\geq 0}^N \mid A^{(Q)}x = b^{(Q)}\}$ with $A^{(Q)} \in \mathbb{Z}^{M^{(Q)} \times N}$. Suppose that this PQ-representation resembles a configuration ILP in the sense that vectors in P are (extended) configurations and that Q contains a job/item vector. Then by solving the convexified relaxation of the corresponding ILP from Lemma 20 in time $2^{O((M^{(Q)}+N)^2)} (M^{(P)} \log(\max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty, \|b^{(P)}\|_\infty, \|b^{(Q)}\|_\infty\} m))^{O(1)}$, we obtain a reduced instance where the number of jobs/items in a configuration is bounded by*

$$((M^{(Q)} + N)M^{(P)} \max\{\|A^{(P)}\|_\infty, \|A^{(Q)}\|_\infty\})^{O((M^{(Q)}+N)M^{(P)})}.$$

C.3 Balancing lemma

Here, we give a proof of the balancing result by Govzmann et al. [19] (Lemma 14). We follow their general proof structure, but notation has been aligned with the rest of this work.

In the following, $L_{\pi,i}$ denotes the total load assigned to some machine i by some schedule π and $x_i^{(\pi)}$ refers to the vector of job multiplicities assigned to machine i by schedule π . For $j \in [d]$, the j^{th} component of $x_i^{(\pi)}$ is denoted by $x_{i,j}^{(\pi)}$.

For machines $i_1, i_2 \in [m]$, schedule π and some job type j , define the *gap* $\Delta_{i_1, i_2}^{\pi, j} = x_{i_1, j}^{(\pi)} - x_{i_2, j}^{(\pi)}$. The *gap size* of π is defined by $\mu(\pi) = \max_{i_1, i_2, j} \Delta_{i_1, i_2}^{\pi, j}$ and the set of *gap constellations* by

$$I(\pi) = \{(i_1, i_2, j) \in [m] \times [m] \times [d] \mid \mu(\pi) = \Delta_{i_1, i_2}^{\pi, j}\}.$$

Govzman et al. [19] show that in an arbitrary schedule where the gap size exceeds p_{\max} , the assignment vectors can be rebalanced by performing a finite sequence of exchanges

between pairs of machines without affecting the load on either of the machines. The result is a schedule with gap size at most p_{\max} . A consequence is that there exists an optimal integral schedule such that the assignment vectors are component-wise close to those of a fractional schedule where all jobs are equally distributed among all machines. The original proof is broken down into a number of steps starting with one important observation.

► **Lemma 51** (Govzmann et al. [19]). *Let $b, n \in \mathbb{N}_{>0}$ with $b \leq n$ and let $S = \{s_1, \dots, s_n\} \subset \mathbb{N}_{>0}$ be a multiset of natural numbers. Then there exists a non-empty subset $S' \subseteq S$ with $|S'| \leq b$ and some $\alpha \in \mathbb{N}_{>0}$ such that $\sum_{s \in S'} s = \alpha b$.*

Proof. For $\ell \in \{0, \dots, b\}$, define $r_\ell = \sum_{k=1}^{\ell} s_k \bmod b$. Because $r_\ell \in \{0, \dots, b-1\}$ for all ℓ , by pigeonhole principle there exist $\gamma \in \mathbb{N}_0, \delta \in \mathbb{N}_{>0}$ with $0 \leq \gamma < \delta \leq b$ such that $r_\gamma = r_\delta$. Therefore, $\sum_{k=\gamma+1}^{\delta} s_k = \alpha b$ for some $\alpha \in \mathbb{N}$. Choosing $S' = \{s_{\gamma+1}, \dots, s_\delta\}$ yields the claim. ◀

The idea is to apply this result to a vector of job multiplicities $z = (z_1, \dots, z_d)$, which can also be described by a multiset with d different elements and n' elements in total for some $d \leq n'$. Provided that the multiset has at least p_j elements, it is possible by Lemma 51 to find a subset of jobs whose processing times sum up to a multiple of p_j . This is expressed by the following lemma.

► **Lemma 52** (Govzmann et al. [19]). *Fix some job type $j \in [d]$ and let $z = (z_1, \dots, z_d)$ be a vector with $\sum_{j' \in D} z_{j'} \geq p_j$. Then there exists a vector $z' \neq \mathbf{0}$ such that*

- $0 \leq z'_{j'} \leq z_{j'}$ for all $j' \in [d]$,
- $\sum_{j' \in [d]} p_{j'} z'_{j'} = \alpha p_j$ for some $\alpha \in \mathbb{N}$ and
- $\sum_{j' \in [d]} z'_{j'} \leq p_j$.

Proof. From z , construct the multiset S containing $z_{j'}$ copies of processing time $p_{j'}$ for each $j' \in [d]$. Set $b = p_j$. Now Lemma 51 can be applied, yielding a non-empty subset $S' \subseteq S$ with $|S'| \leq p_j$ such that $\sum_{p \in S'} p = \alpha b$ for some $\alpha \in \mathbb{N}$. Build a vector z' where $z'_{j'}$ equals the number of copies of $p_{j'}$ in S' for each $j' \in [d]$. As S' contains a subset of the jobs in z , $0 \leq z'_{j'} \leq z_{j'}$ holds for all $j' \in [d]$. The condition $z' \neq \mathbf{0}$ is met because S' is non-empty. $\sum_{j' \in [d]} z'_{j'} \leq p_j$ holds because of $|S'| \leq p_j$. ◀

These insights can now be used for the proof of one of the key results.

► **Lemma 53** (Govzmann et al. [19]). *For $P|\{C_{\max}, C_{\min}, C_{\text{envy}}\}^8$, there exists an optimal schedule π such that the gap size $\mu(\pi)$ is at most p_{\max} .*

Proof. Let π_1 be an optimal schedule with $\mu(\pi_1) > p_{\max}$. Govzman et al. [19] show that π_1 can be transformed into a new schedule π_2 such that

1. $\mu(\pi_2) \leq \mu(\pi_1)$ and
2. if $\mu(\pi_2) = \mu(\pi_1)$ then the set $I(\pi_2)$ of gap constellations is strictly smaller than $I(\pi_1)$.

⁸ The original document includes one more objective, $\sum_{i \in [m]} f(L_i^\pi)$, where f is some convex oracle function evaluating the load on machine $i \in [m]$ for some schedule π .

As $\mu(\pi_1) \leq \max_{j \in [d]} n_j$ and $|I(\pi_1)| \leq m^2 p_{\max}$, the number of such transformation steps needed to obtain the target schedule is bounded. For one transformation step, choose $i_1, i_2 \in [m], j \in [d]$ such that $\Delta_{i_1, i_2}^{\pi_1, j} = x_{i_1, j}^{(\pi_1)} - x_{i_2, j}^{(\pi_1)} = \mu(\pi_1)$. There are two cases to consider:

If $L_{\pi_1, i_1} - L_{\pi_1, i_2} \geq p_j$, then one job of type j can be transferred from i_1 to i_2 . This yields a schedule π_2 with $L_{\pi_2, i_1} = L_{\pi_1, i_1} - p_j$ and $L_{\pi_2, i_2} = L_{\pi_1, i_2} + p_j$. Note that for the objective functions C_{\min}, C_{\max} and C_{envy} , the value of π_2 can only improve or remain unchanged compared to π_1 . Hence, π_2 is still an optimal schedule. Moreover, $\Delta_{i_1, i_2}^{\pi_2, j}$ is strictly smaller than $\Delta_{i_1, i_2}^{\pi_1, j}$. Therefore, $\mu(\pi_2) \leq \mu(\pi_1)$ and if $\mu(\pi_2) = \mu(\pi_1)$, then $|I(\pi_2)| < |I(\pi_1)|$.

In the other case, where $L_{\pi_1, i_1} - L_{\pi_1, i_2} < p_j$, Govzman et al. [19] extract a vector z of jobs which can safely be moved from i_2 to i_1 without widening the gap size. This vector is described by $z_{j'} = \max\{0, x_{i_2, j'}^{(\pi_1)} - x_{i_1, j'}^{(\pi_1)}\}$ for each $j' \in [d]$. In other words, it contains only jobs of those types where i_2 outnumbers i_1 . Next, the following inequality is derived for the processing time of z :

$$\sum_{j' \in [d]} z_{j'} p_{j'} = \sum_{j' \in [d]} \max\{0, x_{i_2, j'}^{(\pi_1)} - x_{i_1, j'}^{(\pi_1)}\} p_{j'} \quad (7)$$

$$= \sum_{j' \in [d] \setminus \{j\}} \max\{0, x_{i_2, j'}^{(\pi_1)} - x_{i_1, j'}^{(\pi_1)}\} p_{j'} + \max\{0, x_{i_2, j}^{(\pi_1)} - x_{i_1, j}^{(\pi_1)}\} p_j \quad (8)$$

$$\geq \sum_{j' \in [d] \setminus \{j\}} (x_{i_2, j'}^{(\pi_1)} - x_{i_1, j'}^{(\pi_1)}) p_{j'} + \max\{0, x_{i_2, j}^{(\pi_1)} - x_{i_1, j}^{(\pi_1)}\} p_j \quad (9)$$

$$= \sum_{j' \in [d] \setminus \{j\}} x_{i_2, j'}^{(\pi_1)} p_{j'} - \sum_{j' \in [d] \setminus \{j\}} x_{i_1, j'}^{(\pi_1)} p_{j'} + \max\{0, x_{i_2, j}^{(\pi_1)} - x_{i_1, j}^{(\pi_1)}\} p_j \quad (10)$$

$$= L_{\pi_1, i_2} - x_{i_2, j}^{(\pi_1)} p_j - (L_{\pi_1, i_1} - x_{i_1, j}^{(\pi_1)} p_j) + \max\{0, x_{i_2, j}^{(\pi_1)} - x_{i_1, j}^{(\pi_1)}\} p_j \quad (11)$$

$$= L_{\pi_1, i_2} - L_{\pi_1, i_1} + (x_{i_1, j}^{(\pi_1)} - x_{i_2, j}^{(\pi_1)}) p_j + \max\{0, x_{i_2, j}^{(\pi_1)} - x_{i_1, j}^{(\pi_1)}\} p_j \quad (12)$$

$$\geq L_{\pi_1, i_2} - L_{\pi_1, i_1} + (x_{i_1, j}^{(\pi_1)} - x_{i_2, j}^{(\pi_1)}) p_j \quad (13)$$

$$\geq L_{\pi_1, i_2} - L_{\pi_1, i_1} + (p_{\max} + 1) p_j \quad (14)$$

$$> -p_j + (p_{\max} + 1) p_j \quad (15)$$

$$= p_{\max} p_j \quad (16)$$

Inequality (14) holds because of the assumption $x_{i_2, j}^{(\pi_1)} - x_{i_1, j}^{(\pi_1)} = \Delta_{i_1, i_2}^{\pi_1, j} = \mu(\pi_1) > p_{\max}$ and Inequality (15) follows from $L_{\pi_1, i_1} - L_{\pi_1, i_2} < p_j$.

The next step is to show that the requirement for Lemma 52 is met:

$$\sum_{j' \in [d]} z_{j'} = \sum_{j' \in [d]} z_{j'} \frac{p_{j'}}{p_{j'}} \geq \sum_{j' \in [d]} z_{j'} \frac{p_{j'}}{p_{\max}} = \frac{1}{p_{\max}} \sum_{j' \in [d]} z_{j'} p_{j'} \geq p_j$$

At this point, Lemma 52 can be applied, yielding a vector $z' \in \mathbb{N}^d$ with $z' \neq \mathbf{0}$ such that $0 \leq z'_{j'} \leq z_{j'}$ for all $j' \in [d]$ and $\sum_{j' \in [d]} p_{j'} z'_{j'} = \alpha p_j$ for some $\alpha \in \mathbb{N}$ and $\sum_{j' \in [d]} z'_{j'} \leq p_j$. This vector represents the set of jobs from i_2 to be swapped with α jobs of type p_j from machine i_1 . Since the load remains unchanged on both machines, π_2 is also an optimal schedule.

Recall Constraints 1 and 2, claiming a gap size of at most $\mu(\pi_1)$ for π_2 and a decreased number of gap constellations in case the gap size did not shrink. Govzman et al. [19] proceed by showing that the swap did not move too many jobs of type j from i_2 to i_1 and thereby

reverse the gap. More formally:

$$\alpha = \frac{1}{p_j} \sum_{j' \in [d]} p_{j'} z'_{j'} \leq \frac{1}{p_j} \sum_{j' \in [d]} p_{\max} z'_{j'} = \frac{p_{\max}}{p_j} \sum_{j' \in [d]} z'_{j'} \leq p_{\max}$$

The last inequality holds because of $\sum_{j' \in [d]} z'_{j'} \leq p_j$. Note that α cannot exceed the number of jobs of type p_j because of $\alpha \leq p_{\max} < \Delta_{i_1, i_2}^{\pi_1, j} = x_{i_1, j}^{(\pi_1)} - x_{i_2, j}^{(\pi_1)} = \mu(\pi_1)$. Now recall that $\Delta_{i_1, i_2}^{\pi_1, j}$ was strictly greater than p_{\max} . For $\Delta_{i_2, i_1}^{\pi_2, j}$, this implies:

$$\begin{aligned} x_{i_2, j}^{(\pi_2)} - x_{i_1, j}^{(\pi_2)} &= x_{i_2, j}^{(\pi_1)} + \alpha - (x_{i_1, j}^{(\pi_1)} - \alpha) = x_{i_2, j}^{(\pi_1)} - x_{i_1, j}^{(\pi_1)} + 2\alpha \\ &< -p_{\max} + 2\alpha \\ &\leq -p_{\max} + 2p_{\max} \\ &= p_{\max} \end{aligned}$$

Furthermore, the number of jobs on i_1 strictly decreases because of $\alpha > 0$. It remains to prove that the number of gap constellations involving other machines did not increase in π_2 . Let $\Delta_{i_3, i_4}^{\pi_2, j'}$ be some gap constellation introduced by the swap. Then either $i_3 = i_1$ and $x_{i_3, j'}^{(\pi_2)} = x_{i_2, j'}^{(\pi_1)}$ or $i_4 = i_2$ and $x_{i_4, j'}^{(\pi_2)} = x_{i_2, j'}^{(\pi_1)}$. In other words, the transformation of π_1 into π_2 only caused some gap constellations to be swapped between i_1 and i_2 instead of creating new ones. This settles the proof of Lemma 53. \blacktriangleleft

Let $\hat{\pi}$ be a fractional assignment where all jobs are equally distributed among all machines, i.e., where each machine gets assigned $\frac{np}{m}$ jobs of processing time p for each $p \in D$. Govzman et al. [19] use this optimal fractional schedule $\hat{\pi}$ as a reference for an optimal integral schedule and establish the following.

► Lemma 54 (Govzmann et al. [19]). *For $P \parallel \{C_{\max}, C_{\min}, C_{\text{envy}}\}$, there exists an optimal schedule π such that $|x_{i, j}^{(\pi_1)} - x_{i, j}^{(\hat{\pi})}| \leq p_{\max}$ for all $j \in [d], i \in [m]$.*

Proof. Consider an optimal integral schedule π with $\mu(\pi) \leq p_{\max}$. Such a schedule exists by Lemma 53. Let $j \in [d]$ and let $i_1, i_2 \in [m]$ be two distinct machines such that $x_{i_1, j}^{(\pi)} \leq x_{i_1, j}^{(\hat{\pi})}$ and $x_{i_2, j}^{(\pi)} \geq x_{i_2, j}^{(\hat{\pi})}$. Note that such a pair of machines exists for each job type. For any machine $i \in [m]$,

$$x_{i, j}^{(\hat{\pi})} - x_{i, j}^{(\pi)} = x_{i, j}^{(\hat{\pi})} - x_{i_2, j}^{(\pi)} + x_{i_2, j}^{(\pi)} - x_{i, j}^{(\pi)} \quad (17)$$

$$\leq x_{i, j}^{(\hat{\pi})} - x_{i_2, j}^{(\pi)} + p_{\max} \quad (18)$$

$$= x_{i_2, j}^{(\hat{\pi})} - x_{i_2, j}^{(\pi)} + p_{\max} \quad (19)$$

$$\leq p_{\max} \quad (20)$$

Inequality (18) follows from Lemma 53. Equality (19) holds because $x_{i', j'}^{(\hat{\pi})} = x_{i'', j'}^{(\hat{\pi})}$ for each $j' \in [d], i', i'' \in [m]$. Analogously:

$$x_{i, j}^{(\pi)} - x_{i, j}^{(\hat{\pi})} = x_{i, j}^{(\pi)} - x_{i_1, j}^{(\pi)} + x_{i_1, j}^{(\pi)} - x_{i, j}^{(\hat{\pi})} \quad (21)$$

$$\leq p_{\max} + x_{i_1, j}^{(\pi)} - x_{i, j}^{(\hat{\pi})} \quad (22)$$

$$= p_{\max} + x_{i_1, j}^{(\pi)} - x_{i_1, j}^{(\hat{\pi})} \quad (23)$$

$$\leq p_{\max} \quad (24)$$

\blacktriangleleft

From these insights, the balancing result can finally be derived.

► **Lemma 14** (Govzmann et al. [19]). *For $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$, there exists a kernel where the number of jobs of a specific type on a specific machine is bounded by $2p_{\max}$. So the load of every machine is bounded by $2p_{\max}^2 d$. The kernelization runs in $O(d)$ time.*

Proof. As in the proof of Lemma 54, consider an optimal integral schedule π with $\mu(\pi) \leq p_{\max}$ and let $i_1, i_2 \in [m]$ be two distinct machines such that $x_{i_1, j}^{(\pi)} \leq x_{i_1, j}^{(\hat{\pi})}$ and $x_{i_2, j}^{(\pi)} \geq x_{i_2, j}^{(\hat{\pi})}$ for some $j \in [d]$. Then by (17) to (20), $x_{i_1, j}^{(\pi)}$ is at least $x_{i_1, j}^{(\hat{\pi})} - p_{\max} = \frac{n_j}{m} - p_{\max}$. Because the jobs are integrally assigned and $\lceil \frac{n_j}{m} \rceil - 1 - p_{\max}$ is strictly smaller than $\frac{n_j}{m} - p_{\max}$, $x_{i_1, j}^{(\pi)}$ must be at least $\lceil \frac{n_j}{m} \rceil - p_{\max}$. For the same reason, $x_{i_2, j}^{(\pi)}$ cannot exceed $\lfloor \frac{n_j}{m} \rfloor + p_{\max}$. This makes it possible to preprocess problem instances $P||\{C_{\max}, C_{\min}, C_{\text{envy}}\}$ by assigning $\lceil \frac{n_j}{m} \rceil - p_{\max}$ jobs of each type to each machine. The outcome is a kernel with at most $2p_{\max}m$ jobs of each job type. Moreover, because of $x_{i_2, j}^{(\pi)} \leq \lfloor \frac{n_j}{m} \rfloor + p_{\max}$, at most $2p_{\max}$ of the remaining jobs of each type $j \in [d]$ need to be assigned to any machine $i \in [m]$. For d job types, each with processing time at most p_{\max} , this limits the load on every machine on the remaining instance to $2p_{\max}^2 d$. The running time of this preprocessing step is $O(d)$. ◀

D Complexity

This section covers lower bounds and a connection between uniform and identical machines. In the context of uniform machines (Q), we have τ different machine types (and hence $m \in \mathbb{N}^\tau$ is a vector), each with its own speed s_i , and processing a job of type j on a machine of type i induces load $\frac{p_j}{s_i}$. So P is the special case of Q where $\tau = 1$.

D.1 Connection Between Uniform and Identical Machines

It is a long standing question whether $P||C_{\max}$ is FPT w.r.t. parameter d (with high-multiplicity encoding). The following result shows that this question is equivalent to asking whether $Q||C_{\max}$ is FPT with parameters d and τ .

► **Theorem 55.** *The following statements are equivalent:*

1. $P||C_{\max}$ is FPT w.r.t. d (in the high-multiplicity setting).
2. $Q||C_{\max}$ is FPT w.r.t. d and τ (in the high-multiplicity setting, for jobs and machines).

Proof. (2. \implies 1.) This direction is trivial: An algorithm with running time $f(d, \tau)\langle I \rangle^{O(1)}$ for $Q||C_{\max}$ solves $P||C_{\max}$ in time $f(d, 1)\langle I \rangle^{O(1)} = f(d)\langle I \rangle^{O(1)}$.

(1. \implies 2.) We give a reduction from $Q||C_{\max}$ to $P||C_{\max}$. For an instance I of $Q||C_{\max}$, consisting of processing times p_1, \dots, p_d , job multiplicities n_1, \dots, n_d , speeds s_1, \dots, s_τ , machine multiplicities m_1, \dots, m_τ and makespan threshold u , we construct an instance I' of $P||C_{\max}$ like this: We set the new makespan threshold to $u' := 2s_{\max}u + 1$. The new processing times are $p_1, \dots, p_d, u' - s_1u, \dots, u' - s_\tau u$, the multiplicities are $n_1, \dots, n_d, m_1, \dots, m_\tau$ and there are $m_1 + \dots + m_\tau$ (identical) machines.

Suppose that the $Q||C_{\max}$ instance I is feasible, i.e., there is a schedule σ with makespan at most u . We construct a schedule σ' for the $P||C_{\max}$ instance I' in the following way: For every machine k of type $i \in [\tau]$ in I , take all the jobs that σ schedules on k and place them on any one of the machines in I' , together with one of the dummy jobs that has processing time $u' - s_i u$. Note that there are as many dummy jobs as there are machines in I , matching the number of machines of each type. We schedule the jobs from all machines this way,

obtaining schedule σ' ; now, since σ is feasible (so each machine has jobs with total processing time at most $s_i u$), the jobs scheduled on some machine $l \in [m_1 + \dots + m_\tau]$ in σ' have total processing time at most $u' - s_i u + s_i u = u'$, if the original machine had type i . So in σ' , each machine has load at most u' and hence I' is feasible.

For the other direction, suppose that the constructed instance I' is feasible, i.e., there is a schedule σ' with makespan at most u' . The key observation is that no two dummy jobs can be scheduled on the same machine, since $u' - s_i u + u' - s_j u \geq 2u' - 2s_{\min} u = 4s_{\max} u + 2 - 2s_{\min} u \geq 2s_{\max} u + 2 > 2s_{\max} u + 1 = u'$ holds for any machine types $i, j \in [\tau]$. Construct a schedule σ for I as follows: For each machine $k \in [m_1 + \dots + m_\tau]$, take all non-dummy jobs that are scheduled on this machine and place them on a machine of type $i \in [\tau]$, where i is the type corresponding to the (singular) dummy job placed on machine k in σ' . Doing this for each machine, we obtain a schedule σ for all jobs. Now, consider some machine k of type $i \in [\tau]$ and the jobs scheduled on it by σ . Their total processing time is at most $u' - (u' - s_i u) = s_i u$, which means that in σ , for every type $i \in [\tau]$, each machine of type i has load at most $s_i u$, and hence every job is completed within time u . So I is feasible.

Now, consider the parameters of the constructed $P||C_{\max}$ instance. We have parameters

- $p'_{\max} = O(s_{\max} u)$,
- $d' = d + \tau$,
- $u' = O(s_{\max} u)$,
- $m' = O(\tau m_{\max})$ and
- $n'_{\max} = \max\{n_{\max}, m_{\max}\}$.

So the encoding length $\langle I' \rangle$ is bounded by:

$$\begin{aligned} \langle I' \rangle &\leq O(\log(m') + d'(\log(p'_{\max}) + \log(n'_{\max})) + \log(u')) \\ &= O(\log(\tau m_{\max}) + (d + \tau)(\log(s_{\max} u) + \log(\max\{n_{\max}, m_{\max}\})) + \log(s_{\max} u)) \\ &= \langle I \rangle^{O(1)} \end{aligned}$$

Now, suppose there exists an algorithm for $P||C_{\max}$ with running time $f(d')\langle I' \rangle^{O(1)}$. Given an instance I of $Q||C_{\max}$, we can construct an instance I' of $P||C_{\max}$ using the above reduction, in time $\langle I \rangle^{O(1)}$. We then apply the algorithm for $P||C_{\max}$ to I' in time $f(d')\langle I' \rangle^{O(1)} = f(d + \tau)\langle I \rangle^{O(1)}$ and transform the solution into a solution for I , in time $\langle I \rangle^{O(1)}$. In total, this yields an algorithm for $Q||C_{\max}$ that runs in time $\langle I \rangle^{O(1)} + f(d + \tau)\langle I \rangle^{O(1)} + \langle I \rangle^{O(1)} = f(d + \tau)\langle I \rangle^{O(1)}$, which concludes the proof. \blacktriangleleft

A similar reduction also works for the objective C_{\min} :

► **Theorem 56.** *The following statements are equivalent:*

1. $P||C_{\min}$ is FPT w.r.t. d (in the high-multiplicity setting).
2. $Q||C_{\min}$ is FPT w.r.t. d and τ (in the high-multiplicity setting, for jobs and machines).

Proof. (2. \implies 1.) This direction is again trivial: An algorithm with running time $f(d, \tau)\langle I \rangle^{O(1)}$ for $Q||C_{\min}$ solves $P||C_{\min}$ in time $f(d, 1)\langle I \rangle^{O(1)} = f(d)\langle I \rangle^{O(1)}$.

(1. \implies 2.) We give a reduction from $Q||C_{\min}$ to $P||C_{\min}$. For an instance I of $Q||C_{\min}$, consisting of processing times p_1, \dots, p_d , job multiplicities n_1, \dots, n_d , speeds s_1, \dots, s_τ , machine multiplicities m_1, \dots, m_τ and load lower bound ℓ , we construct an instance I' of $P||C_{\min}$ like this: Let $S := \sum_{i=1}^d n_i p_i$ be the sum of all processing times in the given instance. We define the new load lower bound as $\ell' := S + s_{\max} \ell + 1$ and the new processing times as $p_1, \dots, p_d, \ell' - s_1 \ell, \dots, \ell' - s_\tau \ell$. The multiplicities are $n_1, \dots, n_d, m_1, \dots, m_\tau$ and there are $m_1 + \dots + m_\tau$ (identical) machines.

Suppose that the $Q||C_{\min}$ instance I is feasible, i.e., there is a schedule σ with load at least ℓ on each machine. We construct a schedule σ' for the $P||C_{\min}$ instance I' in the following way: For every machine k of type $i \in [\tau]$ in I , take all the jobs that σ schedules on k and place them on any one of the machines in I' , together with one of the dummy jobs that has processing time $\ell' - s_i\ell$. Note that there are as many dummy jobs as there are machines in I , matching the number of machines of each type. We schedule the jobs from all machines this way, obtaining schedule σ' ; now, since σ is feasible, the jobs scheduled on some machine $k \in [m_1 + \dots + m_\tau]$ in σ' have total processing time at least $s_i\ell + \ell' - s_i\ell = \ell'$, if the original machine had type i . So in σ' , each machine has load at least ℓ' and hence I' is feasible.

For the other direction, suppose that the constructed instance I' is feasible, i.e., there is a schedule σ' with load at least $\ell' = S + s_{\max}\ell + 1$. The key observation is that there has to be at least one dummy job on each machine, as all non-dummy jobs together have load S , which is strictly smaller than ℓ' . Since there are as many dummy jobs as machines, there has to be exactly one dummy job on each machine. Construct a schedule σ for I as follows: For each machine $k \in [m_1 + \dots + m_\tau]$, take all non-dummy jobs that are scheduled on this machine and place them on a machine of type $i \in [\tau]$, where i is the type corresponding to the (singular) dummy job placed on machine k in σ' . Doing this for each machine, we obtain a schedule σ for all jobs. Now, consider some machine k of type $i \in [\tau]$ and the jobs scheduled on it by σ . Their total processing time is at least $\ell' - (\ell' - s_i\ell) = s_i\ell$, which means that in σ , for every type $i \in [\tau]$, each machine of type i has load at least $s_i\ell$, and hence the time needed to process these jobs is at least ℓ . So I is feasible.

Now, consider the parameters of the constructed $P||C_{\min}$ instance. We have parameters

- $p'_{\max} = S + s_{\max}\ell + 1 = O(np_{\max} + s_{\max}\ell)$,
- $d' = d + \tau$,
- $\ell' = S + s_{\max}\ell + 1 = O(np_{\max} + s_{\max}\ell)$,
- $m' = O(\tau m_{\max})$ and
- $n'_{\max} = \max\{n_{\max}, m_{\max}\}$.

So the encoding length $\langle I' \rangle$ is bounded by:

$$\begin{aligned} \langle I' \rangle &\leq O(\log(m') + d'(\log(p'_{\max}) + \log(n'_{\max})) + \log(\ell')) \\ &= O(\log(\tau m_{\max}) + (d + \tau)(\log(np_{\max} + s_{\max}\ell) + \log(\max\{n_{\max}, m_{\max}\}))) \\ &\quad + \log(np_{\max} + s_{\max}\ell) \\ &= \langle I \rangle^{O(1)} \end{aligned}$$

Now, suppose there exists an algorithm for $P||C_{\min}$ with running time $f(d')\langle I' \rangle^{O(1)}$. Given an instance I of $Q||C_{\min}$, we can construct an instance I' of $P||C_{\min}$ using the above reduction, in time $\langle I \rangle^{O(1)}$. We then apply the algorithm for $P||C_{\min}$ to I' in time $f(d')\langle I' \rangle^{O(1)} = f(d + \tau)\langle I \rangle^{O(1)}$ and transform the solution into a solution for I , in time $\langle I \rangle^{O(1)}$. In total, this yields an algorithm for $Q||C_{\min}$ that runs in time $\langle I \rangle^{O(1)} + f(d + \tau)\langle I \rangle^{O(1)} + \langle I \rangle^{O(1)} = f(d + \tau)\langle I \rangle^{O(1)}$, which concludes the proof. \blacktriangleleft

Note that these reductions also yield algorithms for $Q||\{C_{\max}, C_{\min}\}$ by applying the reduction and then solving the resulting instance with an algorithm for $P||\{C_{\max}, C_{\min}\}$.

For the C_{envy} -objective, the above reductions do not seem to work directly. Having only the one dummy job from the C_{\min} -reduction is not enough, as we cannot upper bound the processing time of jobs coming from machines of a specific speed value s_i . One could fix this by adding another dummy job for each machine (so that there are two dummy jobs on each machine). Making sure that there are exactly two dummy jobs on each machine should be easy by setting the processing times and the thresholds accordingly, but we have to assure

that there are always two dummy jobs of the same type i on the same machine (essentially making it a machine of type i). This seems to be much more complicated.

D.2 Implications of the Reductions by Chen et al.

Chen et al. [6] gave two reductions from 3-SAT to $P||C_{\max}$ to show lower bounds for approximation algorithms:

► **Theorem 57** (First reduction from Chen et al. [6]). *For every $\delta > 0$ with $\frac{1}{\delta}$ integer, 3-SAT' (a slightly restricted variant of 3-SAT that retains the same ETH-hardness as 3-SAT) with N variables can be reduced to $P||C_{\max}$ with:*

- $d = n = O(\frac{N}{\delta}) = O(N)$ jobs (each with its own processing time),
- $m = O(\frac{N}{\delta}) = O(N)$ machines,
- makespan $u = O(2^{\frac{3}{\delta}} N^{1+\delta}) = O(N^{1+\delta})$
- largest processing time $p_{\max} = O(N^{1+\delta})$
- encoding length $\langle I \rangle = O(d \log(p_{\max}) + \log(m)) = O(\frac{N}{\delta} \log(N^{1+\delta}) + \log(\frac{N}{\delta})) = O(N \log(N))$,

assuming that δ is set to some constant.

► **Theorem 58** (Second reduction from Chen et al. [6]). *3-SAT with N variables can be reduced to $P||C_{\max}$ with:*

- An arbitrary number m of machines (this can be freely chosen),
- $n = O(N + m)$ jobs,
- $d = O(N)$ different processing times,
- $n_{\max} = O(m)$ as the highest multiplicity,
- $p_{\max} = 2^{O(\frac{N \log^2(m)}{m})}$ and hence
- encoding length $\langle I \rangle = O(d \log(p_{\max} + n_{\max}) + \log(m)) = O(N \log(2^{O(\frac{N \log^2(m)}{m})} + m) + \log(m)) = O(N \frac{N \log^2(m)}{m} \log(m) + \log(m)) = O(\frac{N^2 \log^3(m)}{m})$.

These reductions can be used to show the following lower bounds:

► **Theorem 2.** *Let $\varepsilon > 0$. Unless the ETH fails, $P||C_{\max}$ cannot be solved in time $\langle I \rangle^{O(d^{1-\varepsilon})} p_{\max}^{O(1)}$, $\langle I \rangle^{O(1)} d^{O(d^{1-\varepsilon})} p_{\max}^{O(1)}$ or $\langle I \rangle^{o(\frac{d}{\log(d)})} p_{\max}^{O(1)}$.*

Proof. Let $c \in \mathbb{R}$ be a constant such that $p_{\max}^{O(1)} = (2^{O(\frac{N \log^2(m)}{m})})^{O(1)} = 2^{c \frac{N \log^2(m)}{m}}$. Then for any constant $\delta > 0$, we can set m large enough that $2^{c \frac{N \log^2(m)}{m}} \leq 2^{\delta N}$. So multiplying $p_{\max}^{O(1)}$ with any of the below terms yields a better-than-ETH running time.

As we set m large enough, we also get $\frac{\log^3(m)}{m} = O(1)$ and hence $\langle I \rangle = O(N^2)$.

1.) We have for some constant c that

$$\langle I \rangle^{O(d^{1-\varepsilon})} = N^{O(N^{1-\varepsilon})} = N^{c N^{1-\varepsilon}} = (2^{\log(N)})^{c N^{1-\varepsilon}} = 2^{o(N)}$$

and such a running time would contradict the ETH.

2.) We have for some constants c_1, c_2, c_3 that

$$\begin{aligned} \langle I \rangle^{O(1)} d^{O(d^{1-\varepsilon})} &= N^{O(1)} O(N)^{O(N)^{1-\varepsilon}} = N^{c_1} (c_2 N)^{(c_3 N)^{1-\varepsilon}} = 2^{c_1 \log(N)} (2^{\log(c_2 N)})^{(c_3 N)^{1-\varepsilon}} \\ &= 2^{c_1 \log(N)} 2^{\log(c_2 N) (c_3 N)^{1-\varepsilon}} \\ &= 2^{o(N)} \end{aligned}$$

and such a running time would contradict the ETH.

3.) We have

$$\langle I \rangle^{o\left(\frac{d}{\log(d)}\right)} = N^{o\left(\frac{N}{\log(N)}\right)} = (2^{\log(N)})^{o\left(\frac{N}{\log(N)}\right)} = 2^{\log(N) o\left(\frac{N}{\log(N)}\right)} = 2^{o(N)}$$

and such a running time would contradict the ETH. \blacktriangleleft

With the first reduction (Theorem 57), one can show the following:

► **Corollary 59.** *Let $\delta > 0$. Then $P||C_{\max}$ cannot be solved in time $2^{O(d+m+l+u+p_{\max}+\langle I \rangle)^{1-\delta}}$ unless the ETH fails.*

► **Corollary 60.** *Let $\delta > 0$. Then $P||C_{\max}$ cannot be approximated with additive error εp_{\max} in time $2^{O(\frac{1}{\varepsilon}+d+m+l+u+p_{\max}+\langle I \rangle)^{1-\delta}}$ unless the ETH fails.*

Proof. Setting $\varepsilon := \frac{1}{p_{\max}+1}$ yields $OPT + \varepsilon p_{\max} = OPT + \frac{1}{p_{\max}+1} p_{\max} < OPT + 1$ and hence for this choice of ε , such an additive approximation computes an optimal solution. So with $\varepsilon := \frac{1}{p_{\max}+1}$, a running time $2^{O(\frac{1}{\varepsilon}+d+m+l+u+p_{\max}+\langle I \rangle)^{1-\delta}}$ would give an exact algorithm with running time $2^{O(d+m+l+u+p_{\max}+\langle I \rangle)^{1-\delta}}$, contradicting the ETH via Corollary 59. \blacktriangleleft

Note that the running time of the additive approximation algorithm is $(mp_{\max})^{O(\frac{1}{\varepsilon})}$. So the following lower bound shows that the dependency on ε and m is almost optimal:

► **Theorem 4.** *Let $\delta > 0$. Then $P||C_{\max}$ cannot be approximated with additive error at most εp_{\max} in time $(mp_{\max})^{O((\frac{1}{\varepsilon})^{1-\delta})}$, unless the ETH fails.*

Proof. Again, setting $\varepsilon := \frac{1}{p_{\max}+1}$ yields $OPT + \varepsilon p_{\max} = OPT + \frac{1}{p_{\max}+1} p_{\max} < OPT + 1$ and hence for this choice of ε , such an additive approximation computes an optimal solution. So with $\varepsilon := \frac{1}{p_{\max}+1}$, a running time $(mp_{\max})^{O((\frac{1}{\varepsilon})^{1-\delta})}$ would give an exact algorithm for 3-SAT with running time $2^{O(\log(NN^{1+\delta'})(N^{1+\delta'})^{1-\delta})} = 2^{o(N)}$ via Theorem 57, contradicting the ETH. \blacktriangleleft

Koutecký and Zink showed the following:

► **Theorem 61** (Koutecký & Zink [31]). *$\{R, Q\}||\{C_{\max}, \sum w_j C_j, \ell_2\}$ can be solved in time $mn^{O(d)}$ via dynamic programming.*

With either reduction from [6], one can show that this running time is essentially tight, even for $P||C_{\max}$:

► **Corollary 62.** *Unless the ETH fails, $P||C_{\max}$ cannot be solved in time $mn^{O(d^{1-\varepsilon})}$ for any $\varepsilon > 0$.*

Proof. If we use the first reduction (Theorem 57), $m = O(N)$ and $n = d = O(N)$. This yields

$$mn^{O(d^{1-\varepsilon})} = NN^{O(N^{1-\varepsilon})} = 2^{o(N)},$$

which contradicts the ETH. Similarly, with the second reduction (Theorem 58), we have some number of machines m that we can freely choose and set to a constant, $n = O(N+m) = O(N)$ jobs and $d = O(N)$ different processing times. This yields

$$mn^{O(d^{1-\varepsilon})} = N^{O(N^{1-\varepsilon})} = 2^{o(N)},$$

which also contradicts the ETH. \blacktriangleleft