New Support Size Bounds for Integer Programming, Applied to Makespan Minimization on Uniformly Related Machines

Sebastian Berndt
Institute for Theoretical Computer Science, University of Lübeck, Germany

Hauke Brinkop
Kiel University, Germany

Klaus Jansen
Kiel University, Germany

Matthias Mnich
Hamburg University of Technology, Institute for Algorithms and Complexity, Germany

Tobias Stamm
Hamburg University of Technology, Institute for Algorithms and Complexity, Germany

Abstract

Mixed-integer linear programming (MILP) is at the core of many advanced algorithms for solving fundamental problems in combinatorial optimization. The complexity of solving MILPs directly correlates with their support size, which is the minimum number of non-zero integer variables in an optimal solution. A hallmark result by Eisenbrand and Shmonin (*Oper. Res. Lett.*, 2006) shows that any feasible integer linear program (ILP) has a solution with support size $s \leq 2m \cdot \log(4m\Delta)$, where m is the number of constraints, and Δ is the largest absolute coefficient in any constraint.

Our main combinatorial result are improved support size bounds for ILPs.

We show that any ILP has a solution with support size $s \leq m \cdot (\log(3A_{\max}) + \sqrt{\log(A_{\max})})$, where $A_{\max} \coloneqq \|A\|_1$ denotes the 1-norm of the constraint matrix A. Furthermore, we show support bounds in the linearized form $s \leq 2m \cdot \log(1.46A_{\max})$. Our upper bounds also hold with $A_{\max}$ replaced by $\sqrt{m}\Delta$, which improves on the previously best constants in the linearized form.

Our main algorithmic result are the fastest known approximation schemes for fundamental scheduling problems, which use the improved support bounds as one ingredient.

We design an efficient approximation scheme (EPTAS) for makespan minimization on uniformly related machines ($Q||C_{\max}$). Our EPTAS yields a $(1+\varepsilon)$-approximation for $Q||C_{\max}$ on N jobs in time $2^{\mathcal{O}(1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon)))} + \mathcal{O}(N)$, which improves over the previously fastest algorithm by Jansen, Klein and Verschae (*Math. Oper. Res.*, 2020) with run time $2^{\mathcal{O}(1/\varepsilon \log^4(1/\varepsilon))} + N^{\mathcal{O}(1)}$. Arguably, our approximation scheme is also simpler than all previous EPTASes for $Q||C_{\max}$, as we reduce the problem to a novel MILP formulation which greatly benefits from the small support.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Integer programming, scheduling algorithms, uniformly related machines, makespan minimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.13

Related Version *arXiv Version*: `https://arxiv.org/abs/2305.08432` [7]

Funding *Hauke Brinkop*: Partially supported by DFG project `JA 612/25-1`, Fein-granulare Komplexität und Algorithmen für Scheduling und Packungen.
Klaus Jansen: Partially supported by DFG project `JA 612/25-1`, Fein-granulare Komplexität und Algorithmen für Scheduling und Packungen.
Matthias Mnich: Partially supported by DFG project `MN 59/4-1`, Multivariate algorithms for high-multiplicity scheduling.

© Sebastian Berndt, Hauke Brinkop, Klaus Jansen, Matthias Mnich, and Tobias Stamm;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The Integer Linear Programming (ILP) problem is to find an optimal integral solution $\boldsymbol{x}^\star \in \mathbb{Z}^n_{\geq 0}$, which minimizes a linear objective $\boldsymbol{c}^\intercal \boldsymbol{x}$ subject to a system $A\boldsymbol{x} = \boldsymbol{b}$ of m linear constraints. In the Mixed-Integer Linear Programming (MILP) problem, the solution sought is a pair $(\boldsymbol{x}^\star, \boldsymbol{y}^\star) \in \mathbb{Z}^n_{\geq 0} \times \mathbb{Q}^r_{\geq 0}$, which minimizes $\boldsymbol{c}^\intercal(\boldsymbol{x}\ \boldsymbol{y})$ subject to $A\boldsymbol{x} + B\boldsymbol{y} = \boldsymbol{b}_1$ and $C\boldsymbol{y} = \boldsymbol{b}_2$. Solving (M)ILPs is at the core of many advanced algorithms for fundamental combinatorial optimization problems. Very often, the run time of these algorithms scales with the *support size* of the (M)ILPs, which is the smallest number of non-zero entries in an optimal solution $\boldsymbol{x}^\star$ resp. $(\boldsymbol{x}^\star, \boldsymbol{y}^\star)$. Thus, support size bounds have found applications all over computer science, for example in scheduling [5, 25], logic [30, 35], and even complexity theory [21]. Therefore, the smaller the support size, the better these results become. Hence, an important research direction is to prove strong upper bounds on the support size of (M)ILPs. The original result on support size bounds, which is already finding its way into the standard curriculum of integer programming courses [37, Lemma 6.1] is:

▶ **Proposition 1** (Eisenbrand, Shmonin [15, Thm. 1(ii)]). *Any feasible and bounded integer program with m constraints admits a solution with support size $s \leq 2m\log(4m\Delta)$, where Δ is the largest absolute value of any entry in the constraint matrix A.*

The result by Eisenbrand and Shmonin is about feasible solutions, and thus does not depend on any objective function. Aliev et al. [2] improved the Eisenbrand-Shmonin bound to $2m\log(2\sqrt{m}\Delta)$, and showed it to hold even for the support size of optimal solutions with respect to any linear objective function. Recently, Gribanov et al. [19] were the first to achieve a leading coefficient of 1, with a bound of the form $m\log(c_1 \cdot \sqrt{m}\Delta \cdot \sqrt{\log(c_2 \cdot \sqrt{m}\Delta)})$, and improved constants of $1.18m\log(7.02\sqrt{m}\Delta)$ in the linearized form. For the special cases of positively space spanning matrices [1], and in the average case over all right-hand sides [34], results on the order of $\mathcal{O}(m)$ have recently been obtained. One important application of the Eisenbrand-Shmonin bound are efficient polynomial-time approximation schemes (EPTAS) for scheduling problems [23]. An EPTAS computes, for any problem instance $\mathcal{I}$ and any $\varepsilon > 0$, a solution whose value is within $(1+\varepsilon)$ of the optimal solution value in time $f(1/\varepsilon)\langle\mathcal{I}\rangle^{\mathcal{O}(1)}$, where $\langle\mathcal{I}\rangle$ denotes the encoding size of $\mathcal{I}$. Several EPTAS were devised for the classical scheduling problem of makespan minimization on uniformly related machines. This problem is denoted as $Q||C_{\max}$ in Graham's 3-field notation [18]. The input to $Q||C_{\max}$ is a set $\mathcal{J}$ of N jobs, each of which is characterized by an integer processing time p_j, and a set $\mathcal{M}$ of M machines, each of which is characterized by an integer speed s_i. The goal is to find an assignment (or schedule) $\sigma : \mathcal{J} \to \mathcal{M}$ of jobs to machines, which minimizes the maximum completion time $C_{\max} := \max_{i\in\mathcal{M}} \sum_{j\in\sigma^{-1}(i)} p_j/s_i$ of any machine. Problem $Q||C_{\max}$ is well-known to be NP-hard, even in the special case of unit speeds $s_1 = \cdots = s_m$, but still approximable to arbitrary precision, in contrast to the setting of unrelated machines. The previously fastest EPTAS for $Q||C_{\max}$ is due to Jansen, Klein and Verschae [25, 26], and runs in time $2^{\mathcal{O}\left(1/\varepsilon \log^4(1/\varepsilon)\right)} + N^{\mathcal{O}(1)}$. Since their result first appeared in 2016, it has been an open question whether the exponential dependency on $1/\varepsilon$ can be improved. In particular, there is a gap to the best-known lower bound, which shows that a run time of $2^{\mathcal{O}\left((1/\varepsilon)^{1-\delta}\right)} + N^{\mathcal{O}(1)}$ is not possible for any $\delta > 0$, assuming the Exponential-Time Hypothesis [12]. It is unknown, whether this gap can be improved to $\delta = 0$, even under stronger assumptions such as Gap-ETH [33]. Further, a gap remains to the best-known run time $2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log(\log(1/\varepsilon)))} + N^{\mathcal{O}(1)}$ for the case of unit speeds $s_1 = \cdots = s_m$ [8]. Our work shows that this gap could be closed with an algorithm for MILPs with few constraints, as efficient as those available for ILPs with few constraints.

Combinatorial results. Our main combinatorial result improves on the fundamental support size bound of Proposition 1 with regard to the new parameter $A_{\max}$. Prior work used the maximum norm $\Delta := \|A\|_{\max} = \max_{i,j} |A_{i,j}|$, the largest absolute value of an entry in the constraint matrix A. Instead, we use the 1-norm $A_{\max} := \|A\|_1 = \max_i \|A_i\|_1$, the largest 1-norm of any column A_i in A. The matrix 1-norm is sub-multiplicative, consistent with the vector 1-norm and recognizes some sparsity, all in contrast to the maximum norm. This makes $A_{\max}$ a natural parameter to consider for studying the properties of ILPs. We show:

▶ **Theorem 2.** *Any feasible bounded ILP with an m-row constraint matrix A with 1-norm $A_{\max}$ has an optimal solution with support size $s \leq m \cdot (\log(3A_{\max}) + \sqrt{\log(A_{\max})})$.*

We also derive parametric support size bounds in linearized form, like $s \leq 1.1m \cdot (3.42A_{\max})$. As all our upper bounds can equally be derived with $A_{\max}$ replaced by $\sqrt{m}\Delta$, we thereby improve on the constants of Gribanov et al. [19] in this form. For the parameter $A_{\max}$, we show an asymptotically matching lower bound on the support size of an optimal solution:

▶ **Theorem 3.** *For any $m \in \mathbb{Z}_{\geq 0}$ and any $A_{\max} \in \mathbb{Z}_{\geq 1}$, there is an ILP with m constraints, $n := m \cdot (\lfloor \log(A_{\max}) \rfloor + 1) \geq m \cdot \log(A_{\max})$ variables, and 1-norm $A_{\max}$ of the constraint matrix, whose unique optimal solution is the $\mathbf{1}$-vector.*

To obtain this lower bound, we adapt the previously best lower bound of $m \log(\Delta)$ on the support size of an optimal solution by Berndt, Jansen and Klein [9].

Algorithmic results. We use our upper bounds on the support sizes of optimal solutions to ILPs from Theorem 2 as *one* ingredient to obtain new algorithmic results. Namely, we design a new EPTAS for $Q||C_{\max}$, which is asymptotically faster than all previous EPTAS for $Q||C_{\max}$. See Table 1 for a survey of prior work on approximation algorithms for $Q||C_{\max}$.

▶ **Theorem 4.** *There is an algorithm for $Q||C_{\max}$ that, for any $\varepsilon > 0$ and any set of N jobs, computes a $(1+\varepsilon)$-approximate schedule in time $2^{\mathcal{O}\left(1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon))\right)} + \mathcal{O}(N)$.*

Table 1 History of selected complexity results for approximating $Q||C_{\max}$.

authors	year	approximation	run time
Gonzales, Ibarra, Sahni [17]	1977	$2 - \frac{2}{M+1}$	$\mathcal{O}(N \log(N))$
Cho, Sahni [13]	1980	$1 + \sqrt{\frac{M-1}{2}}$	$\mathcal{O}(N \log(N))$
Woeginger [38]	1999	$2 - \frac{1}{M}$	$\mathcal{O}(N \log(N))$
Hochbaum, Shmoys [22]	1988	$1+\varepsilon$	$N^{\mathcal{O}(1/\varepsilon^2 \log(1/\varepsilon))}$
Azar, Epstein [4]	1998	$1+\varepsilon$	$N^{\mathcal{O}(1/\varepsilon^2)}$
Jansen [23]	2010	$1+\varepsilon$	$2^{\mathcal{O}(1/\varepsilon^2 \log^3(1/\varepsilon))} + N^{\mathcal{O}(1)}$
Jansen, Robenek [28]	2012	$1+\varepsilon$	$2^{\mathcal{O}(1/\varepsilon^2 \log^3(1/\varepsilon))} + N^{\mathcal{O}(1)}$
Jansen, Klein, Verschae [25, 26]	2016	$1+\varepsilon$	$2^{\mathcal{O}\left(1/\varepsilon \log^4(1/\varepsilon)\right)} + N^{\mathcal{O}(1)}$
This paper	2023	$1+\varepsilon$	$2^{\mathcal{O}\left(1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon))\right)} + \mathcal{O}(N)$

Compared to previous works, we devise a novel MILP formulation for $Q||C_{\max}$ whose constraint matrix has small column norm. Solving this new formulation not only yields the fastest known EPTAS for $Q||C_{\max}$, but also an EPTAS which is conceptually much simpler than all previous ones. In particular, we introduce the following simplifications in our algorithm compared to the previous EPTAS results:

- In contrast to all other previous algorithms, we do not need a case distinction that splits the algorithm and its analysis into three or four different cases, depending on the processing time ratios and numbers of machines. Our algorithm handles all these cases simultaneously by incurring an increase in the constants hidden within the $\mathcal{O}$ terms.
- We set up an MILP where both jobs and machines with similar size and speed are combined into few distinct job and machine classes. All but the longest jobs and machines are then scheduled fractionally. Rounding the fractional allocations for relatively tiny jobs in a machine class was usually done via a separate algorithm by Lenstra, Shmoys and Tardos [32]. Instead, we pack these jobs with a simple greedy strategy.
- We assign relatively huge jobs of each rounded machine speed using basic linear program properties. This creates a linear instead of logarithmic overhead, which we accommodate by increasing the number of integer variables by a constant factor. Previous EPTAS mostly used a complex algorithm by Jansen [24] for a particular bin packing problem.

The only remaining algorithm used as a black-box is the well-known algorithm by Lenstra [31] for solving MILPs with constant dimension (resp. its improvement due to Kannan [29]). To show the versatility of our approach, we applied it to three related scheduling problems. Due to space constraints, the details of these applications are only in the full version [7].

High-Multiplicity Scheduling. We study the problem $Q|HM|C_{\max}$, where both jobs *and* machines are given in the succinct high-multiplicity encoding. We are not aware of any prior constant-factor approximation for $Q|HM|C_{\max}$, only for the restricted setting where only the jobs are given in a high-multiplicity encoding by Filippi and Romanin-Jacur [16].

▶ **Corollary 5.** *There is an algorithm for $Q|HM|C_{\max}$ that, for any $\varepsilon > 0$ and any instance $\mathcal{I}$, computes a $(1+\varepsilon)$-approximate schedule in time $2^{\mathcal{O}(1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon)))} + \langle\mathcal{I}\rangle^{\mathcal{O}(1)}$.*

Few Different Machine Speeds. In the special case of $Q||C_{\max}$ with only k distinct machine speeds, the run time of our algorithm can be improved.

▶ **Theorem 6.** *There is an algorithm for $Q||C_{\max}$ that, for any $\varepsilon > 0$, any set of N jobs and any k distinct machine speeds, computes a $(1+\varepsilon)$-approximate schedule in time $2^{\mathcal{O}(k \cdot 1/\varepsilon \log(1/\varepsilon) \log(\log(1/\varepsilon)))} + \mathcal{O}(N)$.*

Few Different Uniform Machine Types. Jansen and Maack [27] posed $R_K Q||C_{\max}$, a generalization of $Q||C_{\max}$ where each job can have up to K different processing times.

▶ **Theorem 7.** *There is an algorithm for $R_K Q||C_{\max}$ that, for any $\varepsilon > 0$ and any set of N jobs, computes a $(1+\varepsilon)$-approximate schedule in time $2^{\mathcal{O}(K \log(K) 1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon)))} + \mathcal{O}(K \cdot N)$.*

2 Preliminaries

We use $\log(2) = 1$, i.e., base 2 logarithms and denote Euler's number by $e := \exp(1)$. For a vector $\boldsymbol{v}$, let $v_{\min} := \min_\ell v_\ell$ and $v_{\max} := \max_\ell v_\ell$ be its extremal entries and $\operatorname{supp}(\boldsymbol{v}) := \{\ell \mid \boldsymbol{v}_\ell \neq 0\}$ its *support*, i.e., the set of indices with non-zero entries. For an instance $\mathcal{I}$, its *encoding size* $\langle\mathcal{I}\rangle$ is given by $\langle\mathcal{I}\rangle := \sum_{x \in I}(\log(|x|+1)+1)$, the sum over the sizes in binary representations of all quantities. For example, an instance $\mathcal{I}$ of $Q||C_{\max}$ has size $\langle\mathcal{I}\rangle$ logarithmic in the job processing times and machine speeds, but linear in the number of jobs and machines. We generally use i as index for machines, and j as index for jobs.

Mixed-Integer Linear Programs. The set of feasible solutions of any MILP is

$$\mathcal{Q} \coloneqq \left\{ \boldsymbol{x} \in \mathbb{Z}^n_{\geq 0}, \boldsymbol{y} \in \mathbb{R}^r_{\geq 0} \mid \begin{pmatrix} A & B \\ 0 & C \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \boldsymbol{b} \right\} \tag{MILP}$$

for matrices $A \in \mathbb{Z}^{m \times n}$, $B \in \mathbb{Z}^{m \times r}, C \in \mathbb{Z}^{s \times r}$ and a vector $\boldsymbol{b} \in \mathbb{Z}^{m+s}$. The encoding size $\langle \mathit{MILP} \rangle$ of (MILP) is logarithmic in the absolute value $\Delta \coloneqq \max_{i,j} |A_{i,j}|$ of the largest coefficient and right-hand side $\boldsymbol{b}$, but linear in the number of variables and constraints.

We will make use of the following classical result for finding solutions of (MILP), which was proved first by Lenstra [31] and obtained with improved run time by Kannan [29].

▶ **Proposition 8** (Kannan [29]). *For any instance of* (MILP)*, in time* $2^{\mathcal{O}(n \log(n))} \langle \mathit{MILP} \rangle^{\mathcal{O}(1)}$ *one either finds a solution* $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Q}$ *or determines that* $\mathcal{Q} = \emptyset$*.*

In the following, we reproduce two useful lemmata about the structure of (MILP) solutions, which are inherited from its integral and fractional parts.

▶ **Lemma 9.** *For any instance of* (MILP) *and any* $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \mathcal{Q}$*, in time* $\langle \mathit{MILP} \rangle^{\mathcal{O}(1)}$ *we can find* $(\hat{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \in \mathcal{Q}$ *such that* $\tilde{\boldsymbol{y}}$ *is a vertex solution of the following restricted LP:*

$$\left\{ \boldsymbol{y} \in \mathbb{R}^r_{\geq 0} \mid \begin{pmatrix} B \\ C \end{pmatrix} \cdot \boldsymbol{y} = \boldsymbol{b} - \begin{pmatrix} A \\ 0 \end{pmatrix} \cdot \hat{\boldsymbol{x}} \right\} \ . \tag{R-LP}$$

Proof. By assumption, (R-LP) is feasible, as $\hat{\boldsymbol{y}}$ is a solution. With the ellipsoid algorithm [20, Remark 6.5.2] we find a vertex solution $\tilde{\boldsymbol{y}}$ of (R-LP) in polynomial time. ◀

▶ **Lemma 10.** *For any instance of* (MILP) *and any* $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \mathcal{Q}$ *there is some* $(\tilde{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \mathcal{Q}$ *such that* $\tilde{\boldsymbol{x}}$ *has minimum support* $|\operatorname{supp}(\boldsymbol{x})|$ *of all solutions* $\boldsymbol{x}$ *to the restricted ILP*

$$\left\{ \boldsymbol{x} \in \mathbb{Z}^n_{\geq 0} \mid \begin{pmatrix} A \\ 0 \end{pmatrix} \cdot \boldsymbol{x} = \boldsymbol{b} - \begin{pmatrix} B \\ C \end{pmatrix} \cdot \hat{\boldsymbol{y}} \right\} \ . \tag{R-ILP}$$

Proof. By assumption, (R-ILP) is feasible, as $\hat{\boldsymbol{x}}$ is a solution. Hence, it also has a solution $\tilde{\boldsymbol{x}}$ with minimum support size. Then $(\tilde{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \mathcal{Q}$ holds because of $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \mathcal{Q}$ and $A \cdot \hat{\boldsymbol{x}} = A \cdot \tilde{\boldsymbol{x}}$. ◀

Importantly, Lemma 10 implies that *any* support size bound for an ILP can be directly applied to the integer variables of an MILP. One of these applications is an algorithm to solve MILPs with few constraints. This was presented explicitly and analyzed in terms of m and Δ by Rohwedder and Verschae [36, p. 30], see also Dadush et al. [14]. For us, the crucial underlying idea to efficiently solve MILPs is:

▶ **Lemma 11.** *For any instance of* (MILP) *and any* $s \leq n$*, in time* $2^{\mathcal{O}(s \log(n))} \cdot \langle \mathit{MILP} \rangle^{\mathcal{O}(1)}$ *we either find a solution* $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Q}_{\leq s} \coloneqq \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Q} \colon |\operatorname{supp}(\boldsymbol{x})| \leq s\}$ *of bounded support, or determine that* $\mathcal{Q}_{\leq s} = \emptyset$*.*

Proof. We exhaustively try all choices of $\operatorname{supp}(\boldsymbol{x})$, which are $\binom{n}{s} \leq n^s$ candidates, from $|\operatorname{supp}(\boldsymbol{x})| = 0$ up to s. For each choice we restrict (MILP) to $\operatorname{supp}(\boldsymbol{x})$, by fixing all other integer variables to 0. With Proposition 8 we either find a solution with s integral variables, or determine the infeasibility, in time $s^{\mathcal{O}(s)} \langle \mathit{MILP} \rangle^{\mathcal{O}(1)}$. The run time follows from $s \leq n$. ◀

Note that a support size bound s on any solution of (R-ILP) allows us to use Lemma 11 to find a solution with support size s, or decide the infeasibility of the entire (MILP). Lemma 11 directly extends to optimizing a linear objective function; and to finding a non-zero solution of minimum support, which might be of interest for augmentation algorithms.

3 Refined Support Size Bounds for Integer Linear Programs

In this section, we refine the general support size bounds independent of n for integer linear programs. Previous such bounds used as parameters the number of constraints m, and the largest absolute value Δ of an entry in the constraint matrix A. In our MILP formulation for $Q||C_{\max}$, we bound the support size by the *maximum* 1*-norm* of a column vector, denoted by $A_{\max} := \|A\|_1 = \max_{i=1,\dots,n} \|A_i\|_1$. Clearly, $\Delta \leq A_{\max} \leq m \cdot \Delta$ holds, but it also means that support size bounds using only m and Δ are too coarse for some ranges. In this section, we only consider feasible ILPs $\mathcal{L} \neq \emptyset$ with $\operatorname{rank}(A) = m$. Let $S := \operatorname{supp}(\boldsymbol{v})$ be the support of the vertex $\boldsymbol{v}$, and let $s := |S|$ be the size of the support. Our results are based on:

▶ **Proposition 12** (Aliev et al. [2, Thm. 1(2)]). *Any ILP $\mathcal{L}$ with constraint matrix $A \in \mathbb{Z}^{m\times n}$ has an optimal solution $\boldsymbol{v} \in \mathcal{L}$ with support size $s := |\operatorname{supp}(\boldsymbol{v})| \leq m + \log(\sqrt{\det(A \cdot A^T)})$.*

In this form, the determinant of the support size bound can depend on n. The strength of Proposition 12 is the ability to restrict A to the columns with non-zero variables. For our vertex solution $\boldsymbol{v}$ with support $S = \operatorname{supp}(\boldsymbol{v})$, this is exactly A_S, the columns of the variables in the support. Aliev et al. [2] used the inequality $\sqrt{\det(A_S \cdot A_S^T)} \leq (\sqrt{s}\Delta)^m$ to ultimately obtain the support size bound $s \leq 2m \log(2\sqrt{m}\Delta)$ [3, Thm. 1(ii)]. We analyze the term $\det(A_S \cdot A_S^T)$ in the more fine-grained parameter $A_{\max}$ to obtain a tighter bound:

▶ **Lemma 13.** *For any matrix $A_S \in \mathbb{Z}^{m\times s}$ it holds that $\sqrt{\det(A_S \cdot A_S^T)} \leq (\sqrt{s/m} \cdot A_{\max})^m$.*

Proof. The matrix $G := A_S \cdot A_S^T$ is symmetric and positive semi-definite. If G has an eigenvalue of 0 then $\det(G) = 0$ and the inequality holds. We will thus assume that G is positive definite, i.e., all eigenvalues are positive. It is a classical result that for positive definite matrices, the Hadamard inequality can be strengthened to $\det(G) \leq \prod_{i=1}^m G_{i,i}$, the product of the diagonal entries $G_{i,i}$. We refer to a modern presentation by Browne et al. [10, Thm. 2] for this fact. As $G = A_S \cdot A_S^T$, it is sufficient to bound $\varphi(A_S) := \prod_{i=1}^m \sum_{j=1}^s A_{S;i,j}^2$ subject to $\sum_{i=1}^m |A_{S;i,j}| \leq A_{\max}$ for $j = 1, \dots, s$ by $(s/m \cdot A_{\max}^2)^m$ to obtain our result. We will first characterize a matrix A_S such that $\varphi(A_S)$ is maximal and then relate $\varphi(A_S)$ to $(s/m \cdot A_{\max}^2)^m$. As all entries $A_{S;i,j}$ of A_S occur as squares or absolute values in the optimization, we can assume $A_{S;i,j} \geq 0$ in the following. As φ is monotone in each variable, the matrix A_S that maximizes $\varphi(A_S)$ under the condition $\sum_{i=1}^m |A_{S;i,j}| \leq A_{\max}$ will fulfill these constraints with equality, i.e., $\sum_{i=1}^m |A_{S;i,j}| = A_{\max}$ for $j = 1, \dots, s$. Renaming $A_{S;i,j}$ to $x_{i,j}$ thus gives us the optimization problem:

$$\max \prod_{i=1}^{m} \sum_{j=1}^{s} x_{i,j}^2 \quad \text{s.t.} \sum_{i=1}^{m} x_{i,j} = A_{\max} \quad x_{i,j} \geq 0 \quad \text{for } i = 1, \dots, m; j = 1, \dots, s \ .$$

To bound the optimal solutions to this program, we first consider optimal solutions over the same region with objective function $\sum_{i=1}^m \sum_{j=1}^s x_{i,j}^2$. This is a convex objective function over a polyhedral region. By Bauer's maximum principle [6], the maximum is assumed at a vertex and thus has s non-zero variables. Consequently, we have $\sum_{i=1}^m \sum_{j=1}^s x_{i,j}^2 \leq s \cdot A_{\max}^2$. Now, consider the problem of maximizing $\max \prod_{i=1}^m y_i$ with $\sum_{i=1}^m y_i \leq s \cdot A_{\max}^2$ and $y_i \geq 0$. The logarithm is monotone, so we can apply it to the objective, giving $\sum_{i=1}^m \log(y_i)$ instead. For any solution $x_{i,j}^*$ maximizing $\sum_{i=1}^m \sum_{j=1}^s x_{i,j}^2$, we can compute the y_i^* with $y_i^* = \sum_{j=1}^s {x_{i,j}^*}^2$ that will maximize $\sum_{i=1}^m \log(y_i)$. As the logarithm is concave, we can thus maximize $\sum_{i=1}^m \log(y_i)$ by $y_1^* = \dots = y_m^* = s/m \cdot A_{\max}^2$, as we could otherwise improve a solution by re-balancing it. Hence, $\varphi(A_S) \leq (s/m \cdot A_{\max}^2)^m$, which implies our inequality. ◀

Importantly, by substituting $A_{\max}$ with $\sqrt{m}\Delta$, our inequality in Lemma 13 becomes the one used by Aliev et al. [2]. Therefore, any of the following results can also be obtained for $\sqrt{m}\Delta$ instead of $A_{\max}$. Proposition 12 and Lemma 13 imply the essential intermediate result:

▶ **Corollary 14.** *Any ILP $\mathcal{L}$ with constraint matrix $A \in \mathbb{Z}^{m\times n}$ has an optimal solution $\boldsymbol{v} \in \mathcal{L}$ with $|\operatorname{supp}(\boldsymbol{v})| = s$ such that $s/m \leq 1 + \log(\sqrt{s/m} \cdot A_{\max}) = 1 + \log(A_{\max}) + \log(s/m)/2$.*

The proof of Proposition 12 uses Siegel's Lemma, a deep result from transcendental number theory. In contrast, Berndt et al. [9] derive a support size bound of the same form using only elementary methods, but with worse constants. Building on their approach, we also derive a simple combinatoric, but weaker bound similar to Corollary 14 in the extended version [7].

▶ **Lemma 15.** *Any ILP $\mathcal{L}$ with constraint matrix $A \in \mathbb{Z}^{m\times n}$ has an optimal solution $\boldsymbol{v} \in \mathcal{L}$ with $|\operatorname{supp}(\boldsymbol{v})| = s$ such that $s/m \leq 1 + \log(e) + \log(1 + (s/m) \cdot A_{\max})$.*

Unfortunately, both sides of Corollary 14 are still dependent on s. We resolve Corollary 14 for s in two ways, both parametric in the trade-off between constant and super-constant terms. In the first approach, we bound the logarithm by its tangents.

▶ **Lemma 16.** *For any $\alpha > 0$ and $x > 0$, it holds $\log(x) \leq \alpha \cdot x - \log(e) + \log(\log(e)/\alpha)$.*

Proof. At $x = \log(e)/\alpha$, both sides are $\log(\log(e)/\alpha)$, and the derivatives are α. Hence, the affine function of the right-hand side upper bounds the left-hand side, as $\log(x)$ is concave. ◀

This direct approach allows us to give simple and short formulas for the bounds.

▶ **Theorem 17.** *For any $\alpha \in (0, 1)$ there is an optimal solution $\boldsymbol{v}$ of ILP with*

$$s \leq m \cdot \log(\sqrt{2\log(e)/(e \cdot \alpha)} \cdot A_{\max})/(1 - \alpha) \ .$$

Proof. Applying Lemma 16 with 2α to Corollary 14 yields

$$\begin{aligned} s/m &\leq 1 + \log(A_{\max}) + (2\alpha \cdot s/m - \log(e) + \log(\log(e)/(2\alpha)))/2 \\ &\leq \log(2A_{\max}) + \alpha \cdot s/m + \log\left(\sqrt{\frac{\log(e)}{2\alpha}}\right) \leq \log\left(\sqrt{\frac{2\log(e)}{e\cdot\alpha}} \cdot A_{\max}\right) + \alpha \cdot s/m \ . \end{aligned}$$

We subtract $\alpha \cdot s/m$, and multiply by $m/(1 - \alpha)$, which proves the claim for $0 < \alpha < 1$. ◀

For example, we can set $\alpha = 1/2$ or $\alpha = 1/11$ to obtain the bounds $s \leq 2m \cdot \log(1.46 \cdot A_{\max})$ and $s \leq 1.1m \cdot \log(3.42 \cdot A_{\max})$. This approach, however, only gives bounds with coefficient strictly larger than 1 for the leading term. To reduce this to 1, we make use of advanced analytical function techniques, to tighter analyze the inequality of Corollary 14.

▶ **Lemma 18.** *For $s, m, A_{\max} \geq 1$, the inequality in Corollary 14 is equivalent to*

$$s/m \leq -\log(e) \cdot \mathcal{W}_{-1}(-1/(2\log(e)A_{\max}^2))/2,$$

where $\mathcal{W}_{-1}$ is the -1 branch of the Lambert $\mathcal{W}$-function, the inverse function of $x \mapsto xe^x$.

Proof. We substitute s/m by $-\log(e) \cdot y/2$, and rearrange to obtain:

$$\begin{aligned} & -\log(e) \cdot y/2 \leq 1 + \log(A_{\max}) + \log(-\log(e) \cdot y/2)/2 \\ \Leftrightarrow \quad & \log(-\log(e) \cdot y) + \log(e) \cdot y \geq -(2 + 2\log(A_{\max}) - 1) = -1 - 2\log(A_{\max}) \\ \Leftrightarrow \quad & -\log(e) \cdot y \cdot 2^{\log(e)\cdot y} \geq 1/(2 \cdot A_{\max}^2) \Leftrightarrow y \cdot e^y \leq -1/(2\log(e) \cdot A_{\max}^2) \ . \end{aligned}$$

For the right-hand side $z \coloneqq -1/(2\log(e) \cdot A_{\max}^2)$, we have $-1/e \leq -1/(2\log(e)) \leq z \leq 0$. Therefore, the inequality is satisfied exactly when $\mathcal{W}_{-1}(z) \leq y \leq \mathcal{W}_0(z)$ holds, where $\mathcal{W}_k$ are the real branches of the aforementioned Lambert $\mathcal{W}$-function.

Next, we show $y \leq \mathcal{W}_0(z)$ does not restrict any relevant values. For $x \in [-1/e, 0]$, we have $W_0(x) \geq e \cdot x$. Furthermore $s/m \cdot A_{\max}^2 \geq 1 \geq e/4$ holds for the relevant values of $s/m \geq 1$ and $A_{\max} \geq 1$. Therefore $\mathcal{W}_0(z) \geq -e/(2\log(e)A_{\max}^2) \geq -2 \cdot (s/m)/\log(e) = y$. This shows that the positive solutions are only constrained from above, by $\mathcal{W}_{-1}(z)$. Applying the resubstitution of y to its lower bound gives the claimed result. ◀

Now, we apply techniques analogous to Chatzigeorgiou [11] to prove parametric bounds for the $\mathcal{W}_{-1}(z)$ branch, which we optimize for $z \to 0$ instead of $z \to -1/e$.

▶ **Lemma 19.** *For any $\alpha > 0$ and $u \geq 0$, it holds $-\mathcal{W}_{-1}(-e^{-u-1}) \leq u + \sqrt{2\alpha \cdot u} + \alpha - \ln(\alpha)$.*

Proof. Chatzigeorgiou [11] showed for $x = -\mathcal{W}_{-1}(-e^{-u-1}) - 1$ that $g(x) = u$, where $g(x) \coloneqq x - \ln(1+x)$. To show our result, it thus suffices to show that for all $x \in \mathbb{R} > 0$ and some additive term β, only dependent on α, it holds that

$$-\mathcal{W}_{-1}(-e^{-u-1}) = x + 1 \leq g(x) + \sqrt{2\alpha \cdot g(x)} + \beta + 1 = u + \sqrt{2\alpha \cdot u} + \beta + 1 \ .$$

By definition of g, this reduces to showing $f(x) \coloneqq -\ln(1+x) + \sqrt{2\alpha \cdot (x - \ln(1+x))} + \beta \geq 0$. The function f has a unique minimum, since we will show that $f'(x) = 0$ has only one solution and $\lim_{x\to\infty} f(x) = \infty$. Consider the critical condition:

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) = \frac{-1}{1+x} + \frac{(1 - \frac{1}{1+x}) \cdot \alpha}{\sqrt{2\alpha(x - \ln(1+x))}} = 0 \quad \Leftrightarrow \quad \frac{x\alpha}{\sqrt{2\alpha(x - \ln(1+x))}} = 1 \ . \tag{1}$$

We show that Equation 1 has only one solution, a global minimum, because the second derivative of $f(x)$ is always positive, making $f(x)$ monotonously increasing.

$$\frac{\mathrm{d}}{\mathrm{d}x^2} f(x) = \frac{-x(1 - \frac{1}{1+x})\alpha^2}{(2\alpha(x - \ln(1+x)))^{3/2}} + \frac{\alpha}{\sqrt{2\alpha(x - \ln(1+x))}} > 0 \quad \Leftrightarrow$$

$$\frac{-\alpha^2 x^2/(1+x)}{2\alpha(x - \ln(1+x))} + \alpha > 0 \quad \Leftrightarrow \quad \frac{x^2}{1+x} < 2(x - \ln(1+x)) \quad \Leftrightarrow \quad \frac{x(2+x)}{2(1+x)} \geq \ln(1+x)$$

We know Equation 1 holds at the global minimum. Hence, substituting it in the inequality $f(x) \geq 0$ and applying Lemma 16 on $\ln(1+x)$ bounds the minimal value of $f(x)$ by

$$f(x) = -\ln(1+x) + \alpha x + \beta \geq -\alpha(1+x) - \ln(1/\alpha) + 1 + \alpha x + \beta \geq 0 \quad \Leftrightarrow \quad \beta \geq \alpha + \ln(1/\alpha) - 1 \ .$$

We conclude that $-\mathcal{W}_{-1}(-e^{-u-1}) \leq u + \sqrt{2\alpha \cdot u} + \alpha + \ln(1/\alpha)$, as desired. ◀

From Lemma 18 and Lemma 19 we derive our asymptotically tight support size bound.

▶ **Theorem 20.** *For any $\alpha' > 0$ there is an optimal solution $\boldsymbol{v}$ of ILP with*

$$s \leq m \cdot (\log(A_{\max}) + \sqrt{\alpha'(\log(A_{\max}) + 0.05)} + \alpha'/2 + \log(\sqrt{1/\alpha'}) + 1.03) \ .$$

Proof. We need to rewrite the argument $z \coloneqq -1/(2\log(e)A_{\max}^2)$ in Lemma 18 to the form $-e^{-u-1}$ used in Lemma 19. Hence, solving $z = -e^{-u-1}$ gives $u = \ln(2\log(e)A_{\max}^2/e) = 2\ln(A_{\max}) + \ln(2\log(e)/e)$. We now substitute α by $\alpha'/\log(e)$ to get log instead of ln, and through calculation obtain the bound:

$$
\begin{aligned}
s/m &\leq -\log(e)\mathcal{W}_{-1}(z)/2 \leq \log(e)(u+\sqrt{2\alpha\cdot u}+\alpha-\ln(\alpha))/2\\
&\leq \log(e)(u+\sqrt{2\alpha'/\log(e)\cdot u}+\alpha'/\log(e)-\ln(\alpha'/\log(e)))/2\\
&\leq \log(e)u/2+\sqrt{\log(e)\alpha' u/2}+\alpha'/2-\log(\alpha'/\log(e))/2\\
&\leq \log(A_{\max})+\log(2\log(e)/e)/2+\sqrt{\log(e)\alpha' u/2}-\log(\alpha'/\log(e))/2+\alpha'/2\\
&\leq \log(A_{\max})+\sqrt{\alpha'(\log(A_{\max})+\log(2\log(e)/e)/2)}+\alpha'/2+\log(\log(e)\sqrt{2/(e\alpha')}) \ .
\end{aligned}
$$

Inserting numerical values for terms independent of α' and $A_{\max}$ gives the desired result. ◀

For $\alpha'=1$ and $A_{\max}\geq 1$ we obtain the particularly simple bounds

$$
s \leq m\cdot(\log(A_{\max})+\sqrt{\log(A_{\max})+0.05}+1.53) \leq m\cdot(\log(3A_{\max})+\sqrt{\log(A_{\max})}) \ .
$$

This immediately implies our main support size bound:

▶ **Theorem 2.** *Any feasible bounded ILP with an m-row constraint matrix A with 1-norm $A_{\max}$ has an optimal solution with support size $s\leq m\cdot(\log(3A_{\max})+\sqrt{\log(A_{\max})})$.*

In order to understand how tight our bound is, we adapt a construction by Berndt et al. [9] to obtain an asymptotically matching lower bound on the support size:

▶ **Theorem 3.** *For any $m\in\mathbb{Z}_{\geq 0}$ and any $A_{\max}\in\mathbb{Z}_{\geq 1}$, there is an ILP with m constraints, $n\coloneqq m\cdot(\lfloor\log(A_{\max})\rfloor+1)\geq m\cdot\log(A_{\max})$ variables, and 1-norm $A_{\max}$ of the constraint matrix, whose unique optimal solution is the $\mathbf{1}$-vector.*

Proof. With $d\coloneqq\lfloor\log(A_{\max})\rfloor$ we construct an ILP as follows:

$$
\max \begin{pmatrix} 3^0 & \cdots & 3^d & 3^0 & \cdots & 3^d & \cdots & 3^0 & \cdots & 3^d \end{pmatrix}\cdot\boldsymbol{x} \quad \text{s.t.} \quad \boldsymbol{x}\in\mathbb{Z}_{\geq 0}^{m(d+1)},
$$

$$
\begin{pmatrix} 2^0 & \cdots & 2^d & 0 & & & \cdots & & & 0 \\ 0 & \cdots & 0 & 2^0 & \cdots & 2^d & \cdots & 0 & \cdots & 0 \\ & \vdots & & & & & & \ddots & & \\ 0 & & & \cdots & & & 0 & 2^0 & \cdots & 2^d \end{pmatrix}\cdot\boldsymbol{x} = \begin{pmatrix} 2^{d+1}-1 \\ 2^{d+1}-1 \\ \vdots \\ 2^{d+1}-1 \end{pmatrix} \ .
$$

Because of $\sum_{i=0}^{d}2^i=2^{d+1}-1$, the $\mathbf{1}$-vector is a solution. All coefficients are positive. Hence, in any solution the value of the variables with coefficient 2^d must be less than 2. For any other variable x_i, if it is $x_i\geq 2$, we can increase the objective by setting $x_i\coloneqq x_i-2; x_{i+1}\coloneqq x_{i+1}+1$. Since all objective coefficients are positive, the $\mathbf{1}$-vector is the unique optimal solution. ◀

Hence, Theorem 20 is exact in the dominant term. We pose the question, whether there is a support size bound of the form $m\cdot\log(c\cdot A_{\max})$ for some constant c, as an open problem.

4 An Efficient Approximation Scheme for Makespan Minimization on Uniformly Related machines

Our algorithm follows a typical structure of approximation schemes. First, we *preprocess* the input, by discarding jobs and machines which are so short or slow that assigning them naïvely is acceptable. Next, we perform a *binary search* on the makespan, to reduce the optimization problem to a feasibility problem. Then, we *round* the remaining processing times and machine speeds, according to our makespan guess, to make the resulting instance more structured. Now, we *construct* an MILP, whose feasibility is equivalent to the existence of a schedule. Finally, we solve the MILP and *transform* the solution into a schedule.

4.1 Preprocessing

We reduce the number of parameters bounding the instance to the number of jobs N and a constant fraction δ of the approximation guarantee ε. To enforce $N \geq M$ we potentially drop the $M - N$ slowest machines, as there is an optimal solution not assigning them a job.

Step 1: Removing Negligible Machines and Jobs. We remove all machines slower than $\delta \cdot s_{\max}/N$ and all jobs shorter than $\delta \cdot p_{\max}/N$. Compensating for the lost processing times on a longest job and the machine speeds on a fastest machine introduces an approximation error of at most a factor $(1+\delta)$ each. Now we have $p_{\min} > \delta \cdot p_{\max}/N$ and $s_{\min} > \delta \cdot s_{\max}/N$ and the largest ratios of job processing times $p_{\max}/p_{\min} < N/\delta$ and machine speeds $s_{\max}/s_{\min} < N/\delta$ are bounded only by the parameters N and δ.

Step 2: Preround the Inputs. To achieve a linear run time, we preround the machine speeds and processing times to fewer distinct values. These are rerounded again more carefully at every iteration of the binary search, reducing the run time at the cost of a limited accuracy loss. We round every processing time p_j and machine speed s_i down to the next power of $(1 + \delta)$, introducing errors of no more than $(1 + \delta)$ by construction. Let $\tilde{\eta}_j$ be the number of jobs with rounded processing times $\tilde{p}_j$ be and $\tilde{\mu}_i$ be the number of machines with rounded speed $\tilde{s}_j$. Due to step 1, the amount of *distinct* values after rounding is bounded by $\log_{1+\delta}(p_{\max}/(p_{\max}\delta/N)) = \log_{1+\delta}(s_{\max}/(s_{\max}\delta/N)) \in \mathcal{O}(1/\delta \log(1/\delta \cdot N))$. Because our inputs are sorted, the rounding above can be performed in time $\mathcal{O}(N + 1/\delta \log(1/\delta \cdot N))$.

Step 3: Binary Search for the Makespan. We reduce finding the optimal makespan $\mathrm{OPT}(\mathcal{I})$ to successively checking whether a schedule with makespan T is realizable. The processing time of a longest job on a fastest machine is a lower bound: $\mathrm{OPT}(\mathcal{I}) \geq p_{\max}/s_{\max}$. The schedule assigning all jobs to a fastest machine proves $\mathrm{OPT}(\mathcal{I}) \leq \sum_{j=1}^{N} p_i/s_{\max} \leq N \cdot p_{\max}/s_{\max}$. Hence, we can use a binary search for $\mathrm{OPT}(\mathcal{I})$ in the interval $[p_{\max}/s_{\max}, N \cdot p_{\max}/s_{\max}]$ of ratio N. As accuracy up to a factor $(1+\delta)$ is sufficient, we only need to consider integer powers of $(1 + \delta)$. Our binary search therefore adds a factor of $\mathcal{O}(\log_{1+\delta}(N)) = \mathcal{O}(1/\delta \log(N))$ to the run time of the following steps. We denote the current makespan in the binary search by T. This transforms the problem into either finding a schedule with makespan $(1 + \mathcal{O}(\delta)) \cdot T$, or deciding that no schedule with makespan T exists.

Step 4: Rounding Machine Speeds and Job Processing Times. A $(1 + \varepsilon)$-approximate schedule σ satisfies, for each machine i, the equivalent inequalities

$$\sum_{j \in \sigma^{-1}(i)} \frac{p_j}{s_i} \leq (1 + \varepsilon) \cdot T \quad \Leftrightarrow \quad \sum_{j \in \sigma^{-1}(i)} \frac{p_j}{T} \leq (1 + \varepsilon) \cdot s_i \ . \tag{2}$$

With the aforementioned bounds on the makespan we have enforced the descending chain

$$s_{\max} \geq p_{\max}/T \geq p_{\min}/T > \delta \cdot p_{\max}/(N \cdot T) \geq \delta s_{\max}/N^2 \ .$$

Therefore, all relevant quantities – especially the scaled processing times p_j/T – are in the interval $I \coloneqq (\delta \cdot s_{\max}/N^2, s_{\max}]$, whose left and right end are within a ratio N^2/δ. See also Figure 1 for an overview on the relations between the parameters. We now scale each processing time p_j with $1/T$; this yields an instance with scaled processing times $\tilde{p}_j = 1/T \cdot p_j$, equivalent to our original instance by Equation 2. Our goal is now to cover our interval I by as few subintervals as possible. To this end, we adapt an approach by Berndt

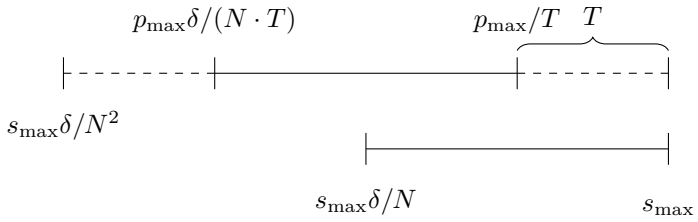

Figure 1 Overview on the range of parameters.

et al. [8] which combines exponential and linear rounding to obtain both sufficient accuracy with few values and useful structural properties. With $\kappa \coloneqq \lceil \log(1/\delta \cdot N^2) \rceil$, consider the points $b_{k,0} \coloneqq s_{\max} \cdot 2^{-k}$ for $k = 0, \dots, \kappa$. The intervals $[b_{k+1,0}, b_{k,0}]$ become exponentially finer, but have a ratio of 2, not the necessary $(1+\delta)$. Therefore, with $\lambda \coloneqq \lceil 1/\delta \rceil$, we add the points $b_{k,\ell} \coloneqq (1 - \ell/(2\lambda)) \cdot b_{k,0}$ for $k = 0, \dots, \kappa - 1$ and $\ell = 0, \dots, \lambda$, which is a linear interpolation between $b_{k,0}$ and $b_{k+1,0}$. Simple calculations show the relations $b_{k,\ell-1}/b_{k,\ell} = 1 + 1/(2\lambda - \ell) \leq 1 + \delta$ and $b_{k+1,0} = b_{k,0}/2 = b_{k,\lceil 1/\delta \rceil}$ for all $k = 0, \dots, \kappa - 1$ and $\ell = 1, \dots, \lambda$, which means we have achieved the necessary precision. Crucially, two jobs within a linear interval of the same parity combine exactly to a job in the next larger linear interval. Formally, this is described by

$$b_{k,\ell} + b_{k,\ell'} = \left(2 - \frac{\ell + \ell'}{2\lambda}\right) \cdot b_{k,0} = \left(1 - \frac{(\ell + \ell')/2}{2\lambda}\right) \cdot b_{k-1,0} = b_{k-1,\frac{\ell+\ell'}{2}} \tag{3}$$

for all $0 \leq \ell \leq \ell' \leq \lambda$ with $\ell + \ell'$ divisible by 2, i.e., ℓ and ℓ' are both odd or both even. Equation 3 will allow us to significantly reduce the number of configurations and the maximum 1-norm of a column. To simplify our notations, we re-index the values of $b_{k,\ell}$ in descending order by setting $b_r \coloneqq b_{k(r),\ell(r)}$ with $k(r) = \lfloor (r-1)/\lambda \rfloor$ and $\ell(r) = (r-1) \bmod \lambda$, such that $b_{k \cdot \lambda + \ell + 1} = b_{k,\ell}$. Therefore, the interval I is covered completely by the $\tau \coloneqq \kappa \cdot \lambda \in \mathcal{O}(1/\delta \log(1/\delta \cdot N))$ intervals $(b_{r+1}, b_r]$ for $r = 1, \dots, \tau$. Finally, we round every machine speed $\tilde{s}_i$ as well as every scaled processing time $\tilde{p}_j$ in $(b_{r+1}, b_r]$ up to b_r. Let μ_i be the number of machines with scaled speed b_i. Let η_j be the number of jobs with scaled processing time b_j. This takes time $\mathcal{O}(1/\delta \log(1/\delta \cdot N))$, which means steps 1 to 4 can be performed in total time $\mathcal{O}(N + 1/\delta \log(N)(1/\delta \log(1/\delta \cdot N) + \tau)) = \mathcal{O}(N + 1/\delta^2 \log^2(1/\delta \cdot N)) \subseteq \mathcal{O}(1/\delta^{2+2\cdot 2}) + \mathcal{O}(N)$.

4.2 Solving an MILP Formulation

Consider the resulting instance after all steps from subsection 4.1 have been applied. Nearly all previous approaches used a mix of *configuration* variables that determine the complete schedule of a machine and *assignment* variables that determine the position of a single job. We combine these different variables into a unified structure called *recursive* configurations. The core idea of our formulation is that an additional machine i of speed b_i can be simulated by placing a *corresponding* job of the same size b_i on a faster machine i' with $b_{i'} < b_i$. In other words, by placing more jobs than the problem requires, we are also allowed to use more machines of the same size than the problem provides. By applying this idea recursively, we can cover a large range of job processing times with configurations of limited range only, as the virtual machines allow us to merge several short jobs (with respect to to a certain machine speed) into a long job. This approach allows us to successively build a configuration from other configurations. Combined with the rounding scheme of Berndt et al. [8] we thereby significantly reduce the overall necessary number of configurations.

For $k = 0, \dots, \kappa$, we define the set $G_k \coloneqq \{r \in \{1, \dots, \tau\} \mid b_r \in [b_{k,0}, b_{k,\lambda}]\}$ of indices of affine slices in our rounding scheme, separated into $G_k^{\text{even}} \coloneqq \{r \in G_k \mid r = 0 \bmod 2\}$ and $G_k^{\text{odd}} \coloneqq G_k \backslash G_k^{\text{even}}$. For $i = 1, \dots, \tau$, we define the set $H_i \coloneqq \{j \in \{1, \dots, \tau\} \mid b_j \in (\delta b_i, b_i]\}$ of indices of *long* job speeds, greater than δb_i and less than the entire machine speed b_i. We will now define *configurations*. All configurations are vectors $\boldsymbol{\gamma}$ with τ entries, each representing multiples of scaled processing times in $\boldsymbol{b} = (b_1, \dots, b_\tau)$. In the following, we describe the configurations for machines with speed b_i. The set $\mathcal{C}_i^{(1)}$ contains all the exact combinations $b_j + b_{j'} = b_i$ as described in Equation 3. With $\boldsymbol{e}_j \in \{0,1\}^\tau$ being the j-th unit vector, let

$$\mathcal{C}_i^{(1)} \coloneqq \{\boldsymbol{e}_j + \boldsymbol{e}_{j'} \mid j, j' \in G_{k(i)-1} \text{ and } j + j' = 2 \cdot (i - \lambda)\} \ .$$

The second set $\mathcal{C}_i^{(2)}$ contains the remaining feasible configurations of long jobs, with at most one job at even and odd positions in an affine slice G_k:

$$\mathcal{C}_i^{(2)} \coloneqq \{\boldsymbol{\gamma} \in \{0,1\}^\tau \mid \boldsymbol{\gamma} \cdot \boldsymbol{b} \leq b_i \text{ and } \operatorname{supp}(\boldsymbol{\gamma}) \subseteq H_i \text{ and for all } k = 0, \dots, \kappa - 1 : \sum_{r \in G_k^{\text{even}}} \gamma_r \leq 1 \text{ and } \sum_{r \in G_k^{\text{odd}}} \gamma_r \leq 1\} \ .$$

Finally, the set of configurations $\mathcal{C}_i$ for machines with speed b_i is defined as $\mathcal{C}_i \coloneqq \mathcal{C}_i^{(1)} \cup \mathcal{C}_i^{(2)}$. The total number of entries in a configuration $\boldsymbol{\gamma}$ is at most $\|\boldsymbol{\gamma}\|_1 \leq 2 \log(2 \cdot 1/\delta)$. Only long jobs from H_i are used, and for each k there is at most one job at an even or odd position in G_k, respectively. We can bound the number of configurations in $\mathcal{C}_i^{(1)}$ by λ^2, and the number of configurations in $\mathcal{C}_i^{(2)}$ by $(\lambda^2)^{2\log(2\cdot 1/\delta)} \in 2^{\mathcal{O}(\log^2(1/\delta))}$, which implies $|\mathcal{C}_i| \in 2^{\mathcal{O}(\log^2(1/\delta))}$.

We require integrality in the variables only for the fastest $L \coloneqq \lambda \lceil \log(1/\delta^3 \log(1/\delta)) \rceil \in \mathcal{O}(1/\delta \log(1/\delta))$ machine speeds. Intuitively, we just need to assign configurations integrally on these machines, as all remaining configurations are very short relative to the machines with the fastest speed. Hence, we can assign them fractionally and round them later on. The overhead from rounding is then scheduled on a fastest machine. The resulting MILP is:

$$\begin{aligned}
\sum_{\boldsymbol{\gamma} \in \mathcal{C}_i} x_{i,\boldsymbol{\gamma}} - \mu_i &\overset{(4)}{=} \sum_{i'=1}^{\tau} \sum_{\boldsymbol{\gamma} \in \mathcal{C}_{i'}} \gamma_i \cdot x_{i',\boldsymbol{\gamma}} - \eta_i \overset{(5)}{\geq} 0 && \text{for } i = 1, \dots, \tau \\
x_{i,\boldsymbol{\gamma}} &\geq 0 && \text{for } i = 1, \dots, \tau;\ \boldsymbol{\gamma} \in \mathcal{C}_i \\
x_{i,\boldsymbol{\gamma}} &\in \mathbb{Z}_{\geq 0} && \text{for } i = 1, \dots, L;\ \boldsymbol{\gamma} \in \mathcal{C}_i.
\end{aligned} \tag{recursive-MILP}$$

Recall that the number of jobs with processing time b_j in configuration $\boldsymbol{\gamma}$ is γ_j, the number of machines with speed b_i is μ_i, and the number of jobs with scaled processing time b_j is η_j. The constraints (4) enforce that the number of additional virtual machines of any speed equals the number of additional corresponding jobs of that same size scheduled somewhere else. The constraints (5) ensure that at least as many jobs of each size are assigned in configurations, as are required by the problem. We show (recursive-MILP) is feasible, up to an approximation factor, for a feasible instance.

▶ **Lemma 21.** *If the original $Q||C_{\max}$-instance $\mathcal{I}$ has a schedule σ with makespan T, then* (recursive-MILP) *is feasible for makespan $(1 + 17\delta) \cdot T$.*

Proof. The schedule σ specifies which jobs are scheduled on any machine i and that those have sufficient speed. We perform steps 1–4, which uses additional speed of at most a factor $(1 + \delta)^7$. If a single job of processing time b_i has been assigned to a machine of speed b_i by σ, then we create a new configuration which assigns this job on that machine, and speed up the

machine by a factor $(1+\delta)$. Otherwise, the difficulty is finding configurations for jobs which are not long, i.e., less than $\delta \cdot b_i$. We repeatedly combine pairs of jobs according to Equation 3 until we have at most one job at an even and odd position in each affine slice G_k on a machine. We partition these jobs j by their value of $\ell = \lfloor \log_\delta(b_j/b_i) \rfloor$, i.e., into slices with ratio δ. The total processing time of jobs in slice ℓ is bounded by $b_i \cdot \delta^\ell \cdot 2 \sum_{j=0}^{\infty} 2^{-j} = 4 \cdot b_i \cdot \delta^\ell$. We iteratively bundle all jobs from the same slice, starting with the last slice until only the first slice is left, into at most 8 configurations of size $b_i \cdot \delta^\ell$. Each configuration can be packed at least half full by the greedy algorithm. The resulting configurations for slice ℓ might have to be rounded in size to the next b_r. Hence, the additional speed introduced for slice ℓ is bounded by $8 \cdot b_i \cdot \delta^\ell \cdot (1+\delta)$. After the creation of such a configuration, if possible, it gets combined again, which uses no additional speed. Eventually, only jobs in the slice for $\ell = 0$ are left. These can be scheduled exactly in a configuration on the machine, by the premise. The additional load incurred on a machine can be bounded by $\sum_{\ell=1}^{\infty} 8 \cdot b_i \cdot \delta^\ell \cdot (1+\delta) = 8 \cdot (1+\delta)/(1-\delta) \cdot \delta \cdot b_i$. For $\delta \leq 1/35$ this is less than $9 \cdot \delta \cdot b_i$ and in total a factor $(1+\delta)^7(1+9\delta) \leq (1+17\delta)$ on the makespan, as claimed. The values $\boldsymbol{x}$ derived from this satisfy constraints (4) and (5) by construction. ◀

For (recursive-MILP), the integer subproblem has $m = 2L \in \mathcal{O}(1/\delta \log(1/\delta))$ constraints and maximal 1-norm of a column $\max \|A_i\|_1 \in \mathcal{O}(\log(1/\delta))$. The first L machine speeds have $L \cdot 2^{\mathcal{O}(\log^2(1/\delta))} = 2^{\mathcal{O}(\log^2(1/\delta))}$ configurations, i.e., integer variables. By Theorem 2 and Lemma 10, if there is a feasible solution of (recursive-MILP), then there is also one with $\mathcal{O}(1/\delta \log(1/\delta) \log(\log(1/\delta)))$ positive integer variables. Lemma 11 thus implies that we can solve (recursive-MILP)(S) in time $2^{\mathcal{O}(1/\delta \log^3(1/\delta) \log(\log(1/\delta)))} \cdot \log(N)^{\mathcal{O}(1)}$, as the encoding size is $\langle (\text{recursive-MILP})(S) \rangle \leq (1/\delta \log(N))^{\mathcal{O}(1)}$. Note that this is the dominant run time in terms of $1/\delta$. We thus either find a feasible solution for the current makespan guess T, or discover that no such solution exists. In the latter case, we discard our current makespan guess and increase it in the next step of the binary search.

4.3 Constructing a Schedule

We need to construct a schedule from a solution $\boldsymbol{x}^\star$ to (recursive-MILP). By constraints (5), we know that $\boldsymbol{x}^\star$ schedules all jobs in some configuration. By constraints (5), we know that $\boldsymbol{x}^\star$ schedules each job in some configuration. By constraints (4), the number of virtual machines equals the additional number of corresponding jobs with equal size scheduled in some configuration. Assigning all configurations to machines within a makespan of approximately T therefore gives a valid schedule, implemented by Algorithm AssignConfsToMachines.

▶ **Lemma 22.** *Algorithm AssignConfsToMachines gives a schedule of makespan at most* $(1+5\delta) \cdot T$ *from a feasible solution* $\boldsymbol{x}^\star$ *to* (recursive-MILP) *in time* $2^{\mathcal{O}(1/\delta)} + \mathcal{O}(N \log^2(N))$.

Proof. The algorithm assigns $\lfloor x_{i,\boldsymbol{\gamma}} \rfloor + 1 \geq x_{i,\boldsymbol{\gamma}}$ configurations, at least as many as $\boldsymbol{x}^\star$ uses. By constraints (5), all jobs get assigned, as they are assigned before the additional jobs corresponding to virtual machines. We always have at least as many virtual machines as $\boldsymbol{x}^\star$, because of the extra configuration added to a fastest machine. We thus need to guarantee that the additional speed scheduled on a fastest machine is sufficiently bounded. The variables of $\boldsymbol{x}^\star$ corresponding to the fastest L machines are already integral. Hence, the first b_i, for which additional speed is put onto the fastest machine, is at most $\delta^3 \cdot s_{\max}/\log(1/\delta)$. There are $\lceil \log(1/\delta) \rceil \lambda + 1$ constraints on the variables $x_{i,\boldsymbol{\gamma}}$ for $\boldsymbol{\gamma} \in \mathcal{C}_i$. Consequently, at most that many variables $x_{i,\boldsymbol{\gamma}}$ can be positive in a vertex solution of the projected LP, which we can find in polynomial time by Lemma 9. As each of these has a size b_i, the total additional speed assigned to a fastest machine is bounded by

AssignConfsToMachines
Input: A feasible solution $\boldsymbol{x}^\star$ to (recursive-MILP).
Output: An approximate schedule σ to the pre-processed instance.

for decreasing machine speeds b_i :
  for each $\boldsymbol{\gamma} \in \mathcal{C}_i$:
    assign $\lfloor x_{i,\boldsymbol{\gamma}} \rfloor$ copies of $\boldsymbol{\gamma}$ to machines with speed b_i
    if $x_{i,\boldsymbol{\gamma}} \notin \mathbb{Z}_{\geq 0}$: assign another copy of $\boldsymbol{\gamma}$ to a fastest machine
    for each processing time b_j used in $\boldsymbol{\gamma}$:
      assign as many jobs of processing time b_j to machines with configuration $\boldsymbol{\gamma}$
      // stop when all jobs are packed or all configurations are filled
      for each job in a configuration $\boldsymbol{\gamma}$ not filled :
        create a virtual machine with the same speed as the corresponding job
return the resulting schedule σ

$$s_{\max}(\lceil \log(1/\delta) \rceil \lambda + 1) \sum_{r=L}^{\infty} b_r = s_{\max}(\lceil \log(1/\delta) \rceil \lambda + 1) \sum_{k=L/\lambda}^{\infty} \sum_{\ell=0}^{\lambda-1} (2 - \ell/\lambda) 2^{-k}$$
$$\leq \frac{\delta^3 \cdot s_{\max}}{\log(1/\delta)} (\lceil \log(1/\delta) \rceil \lambda + 1)(1 + 3\lambda) < 5\delta s_{\max} \ .$$

The last inequality holds for $\delta \leq 1/35$. Adding this much speed to a fastest machine results in a schedule with makespan at most $(1 + 5\delta) \cdot T$. The run time needed to construct the schedule is bounded by the number of machines times the effort per machine, resulting in

$$\mathcal{O}(N \cdot \tau) = \mathcal{O}(1/\delta \cdot N \log(1/\delta \cdot N^2)) \subseteq 2^{\mathcal{O}(1/\delta)} + \mathcal{O}(N \log^2(N)) \ . \quad \blacktriangleleft$$

5 Faster Schedule Construction

The results of the previous section already give us an EPTAS for $Q||C_{\max}$ with almost linear run time of $2^{\mathcal{O}(1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon)))} + \mathcal{O}(N \log^2(N))$. Interestingly, the bottleneck (with respect to N) of this approach is the transformation from a valid MILP solution into a feasible schedule. In this section, we give a more conventional $Q||C_{\max}$ MILP formulation (hybrid-MILP) using both configuration and assignment variables to improve the run time in N. First, we give an algorithm to transform a solution of (recursive-MILP) into a solution of (hybrid-MILP) in sublinear run time in N. Then, we show how to construct a schedule from a solution to (hybrid-MILP) in linear time. This allows us to transform a solution of (recursive-MILP) into a valid schedule in linear run time in N.

Note that Lemma 21 constructs a solution to (recursive-MILP) from a schedule to $Q||C_{\max}$. Hence, both formulations are equivalent up to a multiplicative error of $1 + \mathcal{O}(\varepsilon)$.

5.1 The Hybrid-MILP Formulation

Let $\mathcal{C}'_i := \{\gamma \in \mathbb{N}^\tau \mid \boldsymbol{\gamma} \cdot \boldsymbol{b} \leq b_i \text{ and } \operatorname{supp}(\boldsymbol{\gamma}) \subseteq H_i\}$ be the set of configurations of long jobs with range $1/\delta$ for machine i. For any machine speed b_i and corresponding configuration $\gamma \in \mathcal{C}'_i$, let $\operatorname{free}(i, \gamma) := b_i - \boldsymbol{\gamma} \cdot \boldsymbol{b}$ be the speed of the machine that is free after placing the jobs specified by γ. Then (hybrid-MILP) is given by

$$\sum_{\boldsymbol{\gamma}\in\mathcal{C}'_i} x_{i,\boldsymbol{\gamma}} = \mu_i \qquad \text{for } i = 1,\dots,\tau \tag{6}$$

$$\sum_{i=1}^{\tau}\sum_{\boldsymbol{\gamma}\in\mathcal{C}'_i} \gamma_j \cdot x_{i,\boldsymbol{\gamma}} + \sum_{i=1}^{\tau} y_{i,j} = \eta_j \qquad \text{for } j = 1,\dots,\tau \tag{7}$$

$$\sum_{\boldsymbol{\gamma}\in\mathcal{C}'_i} \text{free}(i,\boldsymbol{\gamma}) \cdot x_{i,\boldsymbol{\gamma}} - \sum_{j=1}^{\tau} b_j \cdot y_{i,j} \geq 0 \qquad \text{for } i = 1,\dots,\tau \tag{8}$$

$$\begin{aligned} x_{i,\boldsymbol{\gamma}},\ y_{i,j} &\geq 0 && \text{for } i = 1,\dots,\tau\,; j = 1,\dots,\tau\,; \gamma \in \mathcal{C}_i \\ y_{i,j} &= 0 && \text{for } i = 1,\dots,\tau\,; j \in \{\min(H_i),\dots,\tau\} \\ x_{i,\boldsymbol{\gamma}} &\in \mathbb{Z}_{\geq 0} && \text{for } i = 1,\dots,L\,; \gamma \in \mathcal{C}_i\ . \end{aligned}$$

(hybrid-MILP)

In this formulation, there are no recursive configurations. Instead, we use configuration variables $x_{i,\gamma}$, indicating how often a configuration γ is used on machine i. Short jobs, taking up speed less than δb_i on machine i, are handled via assignment variables $y_{i,j}$ indicating how many jobs of size b_j are assigned to machines of speed b_i. The constraints (6) enforce that every machine is assigned a configuration, the constraints (7) guarantee that every job is scheduled somewhere, and the constraints (8) make sure that the speed used by short jobs is at most the speed left free by configurations.

We now convert a solution $\boldsymbol{x}^\star$ of (recursive-MILP) into a solution $(\boldsymbol{x}, \boldsymbol{y})$ of (hybrid-MILP).

ConvertMILPsolution
Input: A feasible solution $\boldsymbol{x}^\star$ to (recursive-MILP).
Output: A feasible solution $\boldsymbol{x}, \boldsymbol{y}$ to (hybrid-MILP).

for $i = 1,\dots,\tau$:
  initialize $\eta'_i := \eta_i$ and $x_{i,\gamma} = x^\star_{i,\gamma}$ for $\gamma \in \mathcal{C}_i$, otherwise $x_{i,\gamma} = 0$
for decreasing machine speeds b_i :
  decrease arbitrary $x_{i,\gamma} > 0$ until $\sum_{\gamma\in\mathcal{C}'_i} x_{i,\gamma} = \mu_i$
  do count the number of jobs ζ_j in configurations $x_{i,\gamma}$ for $j \in H_i$
    for every job size b_j with $\zeta_j \geq \eta'_j$:
      substitute $\zeta_j - \eta'_j$ many jobs in appropriate $x_{i,\gamma}$ with $x_{j,\gamma'}$
      (that is, decrease $x_{j,\gamma'}, x_{i,\gamma}$ and increase $x_{i,\gamma''}$, where γ'' is γ with
        jobs j replaced with γ' and jobs shorter than δb_i in γ' dropped)
  until there are no more $\zeta_j \geq \eta'_j$
  decrease every η'_i by ζ_j
for increasing machine speeds b_i :
  calculate the free machine speed $z_i := \sum_{\gamma\in\mathcal{C}'_i} \text{free}(i,\gamma) \cdot x_{i,\gamma}$
  starting with the shortest jobs b_j, increase $y_{i,j}$, decrease z_i and η'_j
  until z_i is 0, $\eta'_j = 0$ or $j \in H_i$
return $(\boldsymbol{x}, \boldsymbol{y})$

▶ **Lemma 23.** *Algorithm ConvertMILPsolution converts a solution $\boldsymbol{x}^\star$ of* (recursive-MILP) *into a solution $(\boldsymbol{x}, \boldsymbol{y})$ of* (hybrid-MILP) *in time $2^{\mathcal{O}(1/\delta \log^2(1/\delta))} \log^{\mathcal{O}(1)}(N)$.*

Proof. The algorithm guarantees that the sum of the configuration variables $x_{i,\gamma}$ for machines with speed b_i is exactly μ_i. Hence, constraints (6) are satisfied. Jobs are only replaced once all original jobs have been accommodated and, by constraints (4), there are exactly as many additional virtual machines, as there are additional jobs. Therefore, this step never reduces the total number of configurations for any machine speed below μ_i. After the loop of lines 3-11 on machines with speed b_i, only jobs of size $b_j \in (\delta b_i, b_i]$, or in other words $j \in H_i$, are assigned via configurations. Additional short jobs would have been assigned via the recursive configurations on these machines, which we neglect by stopping at δb_i. Thus, these machines have sufficient speed to handle these short jobs, which would have been assigned to them. Instead of their original order, we assign the short jobs by increasing size. This does not change the total load assigned, which therefore still remains sufficient. As the sorting ensures that any short job assigned is smaller or equal to some short job that would have been assigned by the original order, we do not assign jobs which are no longer tiny. Due to having sufficient machine speed and not dropping jobs anywhere in the algorithm, we assign all real jobs and hence satisfy constraints (7). Finally, constraints (8) are satisfied by construction, as we never overfill the available speed. The number of configurations $|\mathcal{C}_i'|$ is bounded by $2^{\mathcal{O}(1/\delta \log^2(1/\delta))}$ and all other quantities are bounded by $\mathcal{O}(1/\delta \log(1/\delta \cdot N))$. That allows us to analyze the run time of the algorithm to be within the claimed complexity. ◀

5.2 Constructing a Schedule

Now, we have an assignment for all small jobs, albeit with fractional variables. However, we can assign the small jobs integrally faster than if we had to resolve recursive configurations.

▶ **Lemma 24.** *Given a feasible solution of* (hybrid-MILP)*, a schedule with makespan at most $(1 + 9\delta)T$ can be constructed in time $2^{\mathcal{O}(1/\delta \log^2(1/\delta))} + \mathcal{O}(N)$.*

Proof. For the configuration variables, we pursue the same strategy as in Lemma 22, that is, rounding them down and assigning one configuration to a fastest machine for every rounded variable. Through the use of basic solutions, we construct a schedule introducing a multiplicative error of at most$(1 + 5\delta)$ in comparison to the optimal makespan. The process takes time $2^{\mathcal{O}(1/\delta \log^2(1/\delta))} \log^{\mathcal{O}(1)}(N)$ with our increased number of configurations.

For the assignment variables of the short jobs, we first note that any machine speed b_i is assigned at most 2 fractional short jobs, as a variable only becomes fractional when the preceding or current group runs out of speed. As in Lemma 23, we first sort the assignment variables, in time $\mathcal{O}((1/\delta \log(1/\delta \cdot N))^4)$. By increasing every machine speed by a factor of $(1 + 2\delta)$, those two fractional jobs can be placed on an arbitrary machine of speed b_i, without exceeding the speed. We get another overhead of a factor of $(1 + \delta)$ by greedily packing the assigned jobs to machines, overpacking each machine just slightly. This greedy packing takes time $\mathcal{O}(N + (1/\delta \log(1/\delta \cdot N))^2)$. In total the approximation error is bounded by $(1 + 5\delta)(1 + 2\delta)(1 + \delta) \leq (1 + 9\delta)$ for $\delta < 1/35$, as claimed. ◀

By Lemma 21 for an instance $\mathcal{I}$ with makespan $\mathrm{OPT}(\mathcal{I})$ the recursive-MILP with makespan $(1 + 17\delta) \cdot \mathrm{OPT}(\mathcal{I})$ is feasible. We then converted a solution to one of hybrid-MILP with Lemma 23. A solution for hybrid-MILP with makespan $(1 + 17\delta) \cdot \mathrm{OPT}(\mathcal{I})$ then gives a schedule with makespan $(1 + 9\delta)(1 + 17\delta) \cdot \mathrm{OPT}(\mathcal{I})$ by Lemma 24. Hence, for $\varepsilon \leq 1$ we can pick $\delta = \varepsilon/35$ and obtain a schedule with makespan $(1 + \varepsilon)\,\mathrm{OPT}(\mathcal{I})$. All of the above steps take linear run time in N. Therefore we have achieved Theorem 4.

References

1 Iskander Aliev, Gennadiy Averkov, Jesús A. De Loera, and Timm Oertel. Sparse representation of vectors in lattices and semigroups. *Math. Program.*, 192(1-2, Ser. B):519–546, 2022. doi:10.1007/s10107-021-01657-8.

2 Iskander Aliev, Jesús A. De Loera, Friedrich Eisenbrand, T. Oertel, and Robert Weismantel. The support of integer optimal solutions. *SIAM J. Optim.*, 28(3):2152–2157, 2018. doi:10.1137/17M1162792.

3 Iskander Aliev, Jesús A. De Loera, Timm Oertel, and Christopher O'Neill. Sparse solutions of linear Diophantine equations. *SIAM J. Appl. Algebra Geom.*, 1(1):239–253, 2017. doi:10.1137/16M1083876.

4 Yossi Azar and Leah Epstein. Approximation schemes for covering and scheduling in related machines. *Proc. APPROX 1998*, 1444:39–47, 1998. doi:10.1007/BFb0053962.

5 Nikhil Bansal, Tim Oosterwijk, Tjark Vredeveld, and Ruben van der Zwaan. Approximating vector scheduling: almost matching upper and lower bounds. *Algorithmica*, 76(4):1077–1096, 2016. doi:10.1007/s00453-016-0116-0.

6 Heinz Bauer. Minimalstellen von Funktionen und Extremalpunkte. II. *Arch. Math.*, 11:200–205, 1960. doi:10.1007/BF01236933.

7 Sebastian Berndt, Hauke Brinkop, Klaus Jansen, Matthias Mnich, and Tobias Stamm. New support size bounds for integer programming, applied to makespan minimization on uniformly related machines, 2023. arXiv:2305.08432.

8 Sebastian Berndt, Max A. Deppert, Klaus Jansen, and Lars Rohwedder. Load balancing: The long road from theory to practice. In *Proc. ALENEX 2022*, pages 104–116, 2022. doi:10.1137/1.9781611977042.9.

9 Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. New bounds for the vertices of the integer hull. *Proc. SODA 2021*, pages 25–36, 2021. doi:10.1137/1.9781611976496.3.

10 Patrick Browne, Ronan Egan, Fintan Hegarty, and Padraig Ó Catháin. A survey of the Hadamard maximal determinant problem. *Electron. J. Combin.*, 28(4):Paper No. 4.41,35, 2021. doi:10.37236/10367.

11 Ioannis Chatzigeorgiou. Bounds on the Lambert function and their application to the outage analysis of user cooperation. *IEEE Comm. Lett.*, 17(8):1505–1508, 2013. doi:10.1109/LCOMM.2013.070113.130972.

12 Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *J. Comput. Syst. Sci.*, 96:1–32, 2018. doi:10.1016/j.jcss.2018.03.005.

13 Yookun Cho and Sartaj Sahni. Bounds for list schedules on uniform processors. *SIAM J. Comput.*, 9(1):91–103, 1980. doi:10.1137/0209007.

14 Daniel Dadush, Arthur Léonard, Lars Rohwedder, and José Verschae. Optimizing low dimensional functions over the integers. In *Integer Programming and Combinatorial Optimization*, pages 115–126, 2023. doi:10.1007/978-3-031-32726-1_9.

15 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006. doi:10.1016/j.orl.2005.09.008.

16 Carlo Filippi and Giorgio Romanin-Jacur. Exact and approximate algorithms for high-multiplicity parallel machine scheduling. *J. Sched.*, 12(5):529–541, 2009. doi:10.1007/s10951-009-0122-z.

17 Teofilo Gonzalez, Oscar H. Ibarra, and Sartaj Sahni. Bounds for LPT schedules on uniform processors. *SIAM J. Comput.*, 6(1):155–166, 1977. doi:10.1137/0206013.

18 Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, 5:287–326, 1979. doi:10.1016/S0167-5060(08)70356-X.

19 Dmitry Gribanov, Ivan Shumilov, Dmitry Malyshev, and Panos Pardalos. On Δ-modular integer linear problems in the canonical form and equivalent problems. *J. Glob. Optim.*, pages 1–61, 2022. doi:10.1007/s10898-022-01165-9.

20 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics: Study and Research Texts*. Springer Berlin, Heidelberg, 1988. doi:10.1007/978-3-642-97881-4.

21 Christoph Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. *Proc. LICS 2019*, pages 1–14, 2019. doi:10.1109/LICS.2019.8785850.

22 Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988. doi:10.1137/0217033.

23 Klaus Jansen. An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. *SIAM J. Discrete Math.*, 24(2):457–485, 2010. doi:10.1137/090749451.

24 Klaus Jansen. A fast approximation scheme for the multiple knapsack problem. In *Proc. SOFSEM 2012*, volume 7147 of *Lecture Notes Comput. Sci.*, pages 313–324, 2012. doi:10.1007/978-3-642-27660-6_26.

25 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *Proc. ICALP 2016*, volume 55 of *Leibniz Int. Proc. Informatics*, pages Art. No. 72,13, 2016. doi:10.4230/LIPIcs.ICALP.2016.72.

26 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Math. Oper. Res.*, 45(4):1371–1392, 2020. doi:10.1287/moor.2019.1036.

27 Klaus Jansen and Marten Maack. An EPTAS for scheduling on unrelated machines of few different types. *Algorithmica*, 81(10):4134–4164, 2019. doi:10.1007/s00453-019-00581-w.

28 Klaus Jansen and Christina Robenek. Scheduling jobs on identical and uniform processors revisited. In *Proc. WAOA 2011*, volume 7164 of *Lecture Notes Comput. Sci.*, pages 109–122, 2012. doi:10.1007/978-3-642-29116-6_10.

29 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.

30 Viktor Kuncak and Martin Rinard. Towards efficient satisfiability checking for Boolean algebra with Presburger arithmetic. In *Proc. CADE 2021*, volume 4603 of *Lecture Notes Comput. Sci.*, pages 215–230, 2007. doi:10.1007/978-3-540-73595-3_15.

31 Hendrik W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

32 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming*, 46(3, (Ser. A)):259–271, 1990. doi:10.1007/BF01585745.

33 Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense CSPs. *Proc. ICALP 2017*, 80:Art. No. 78,15, 2017. doi:10.4230/LIPIcs.ICALP.2017.78.

34 Timm Oertel, Joseph Paat, and Robert Weismantel. Sparsity of integer solutions in the average case. *Proc. IPCO 2019*, 11480:341–353, 2019. doi:10.1007/978-3-030-17953-3_26.

35 Ian Pratt-Hartmann. On the computational complexity of the numerically definite syllogistic and related logics. *Bull. Symbolic Logic*, 14(1):1–28, 2008. doi:10.2178/bsl/1208358842.

36 Lars Rohwedder. *Algorithms for Integer Programming and Allocation*. phdthesis, Universität Kiel, 2019. URL: https://macau.uni-kiel.de/receive/diss_mods_00026125.

37 Thomas Rothvoss. Integer optimization and lattices, 2016. Lecture Notes. URL: https://sites.math.washington.edu/~rothvoss/lecturenotes/IntOpt-and-Lattices.pdf.

38 Gerhard J. Woeginger. A comment on scheduling on uniform machines under chain-type precedence constraints. *Oper. Res. Lett.*, 26(3):107–109, 2000. doi:10.1016/S0167-6377(99)00076-0.