Parameterized and Approximation Results for Scheduling with a Low Rank Processing Time Matrix*

Lin Chen[1], Dániel Marx[†2], Deshi Ye[‡3], and Guochuan Zhang[§4]

1 **Department of Computer Science, University of Houston, Houston, TX, USA**
chenlin198662@gmail.com
2 **MTA SZTAKI, Hungarian Academy of Science, Budapest, Hungary**
dmarx@cs.bme.hu
3 **Zhejiang University, College of Computer Science, Hangzhou, China**
yedeshi@zju.edu.cn
4 **Zhejiang University, College of Computer Science, Hangzhou, China**
zgc@zju.edu.cn

Abstract

We study approximation and parameterized algorithms for $R||C_{max}$, focusing on the problem when the rank of the matrix formed by job processing times is small. Bhaskara et al. [2] initiated the study of approximation algorithms with respect to the rank, showing that $R||C_{max}$ admits a QPTAS (Quasi-polynomial time approximation scheme) when the rank is 2, and becomes APX-hard when the rank is 4.

We continue this line of research. We prove that $R||C_{max}$ is APX-hard even if the rank is 3, resolving an open problem in [2]. We then show that $R||C_{max}$ is FPT parameterized by the rank and the largest job processing time p_{max}. This generalizes the parameterized results on $P||C_{max}$ [17] and $R||C_{max}$ with few different types of machines [15]. We also provide nearly tight lower bounds under Exponential Time Hypothesis which suggests that the running time of the FPT algorithm is unlikely to be improved significantly.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases APX-hardness, Parameterized algorithm, Scheduling, Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.STACS.2017.22

1 Introduction

We consider the classical problem of scheduling independent jobs on parallel machines. In this problem, every job j is required to be processed non-preemptively on one of the machines, and has a processing time $p_{ij} \in \mathbb{N}$ if it is processed on machine i. The goal is to assign jobs to machines such that the makespan (maximum job completion time) is minimized.

* A full version of the paper is available at https://www.researchgate.net/publication/313852592_Parameterized_and_approximation_results_for_scheduling_with_a_low_rank_processing_time_matrix.

† Research supported in part by ERC Grant Agreement no. 280152
‡ Research supported in part by NSFC 11271325 and NSFC 11671355.
§ Research supported in part by NSFC 11271325 and NSFC 11671355.

This problem is usually referred to as unrelated machine scheduling (with the objective of makespan minimization), and denoted as $R||C_{max}$. Specifically, if $p_{ij} = p_j/s_i$, the problem is called uniformly related machine scheduling, and denoted as $Q||C_{max}$. Furthermore, if $p_{ij} = p_j$, the problem is called identical machine scheduling and denoted as $P||C_{max}$.

As we will provide details later, the unrelated machine scheduling problem $R||C_{max}$ is considerably harder than its special cases $Q||C_{max}$ and $P||C_{max}$. From the perspective of approximation algorithms, $Q||C_{max}$ admits a PTAS (Polynomial Time Approximation Scheme) [11], while a $(1.5-\epsilon)$-approximation algorithm with $\epsilon > 0$ being any small constant for $R||C_{max}$ implies $P = NP$ [16]. From the perspective of FPT (Fixed Parameter Tractable) algorithms, $P||C_{max}$ and $Q||C_{max}$ are FPT parameterized by p_{max} (the largest job processing time) [15, 17], while $R||C_{max}$ remains NP-hard even if p_{max} is 3 [16]. Consequently, various intermediate models are studied in literature, aiming to bridge the way from $P||C_{max}$ or $Q||C_{max}$ to $R||C_{max}$. Recently, Bhaskara et al. studied the scheduling problem from a new perspective. In their seminal paper [2], they consider the rank of the matrix formed by the processing times of jobs, i.e., the rank of $P = (p_{ij})_{m\times n}$ where m, n are the number of machines and jobs, respectively. From this point of view, $Q||C_{max}$ is the scheduling problem with a matrix of rank 1, while $R||C_{max}$ is the scheduling problem with a matrix of an arbitrary rank, specifically, the rank may be as large as m. It thus becomes a very natural question that whether we can find better algorithms for $R||C_{max}$ if the rank is small.

For simplicity, from now on we call the problem of minimum makespan scheduling with the matrix of processing times that has the rank of d as rank-d scheduling. It is shown by Bhaskara et al. [2] that rank-2 scheduling admits a QPTAS (Quasi-polynomial Time Approximation Scheme), while rank-4 scheduling becomes APX-hard, leaving open the approximability of rank-3 scheduling.

We continue this line of research in this paper by studying approximation and parameterized algorithms for $R||C_{max}$ with respect to the rank of the matrix. Our first result is the following theorem, which answers the open problem in [2].

▶ **Theorem 1.** *Assuming $P \neq NP$, for any fixed $\rho < 2^{-14}$ there does not exist a $(1+\rho)$-approximation algorithm for $R||C_{max}$, even if the rank of the matrix formed by job processing times is 3.*

In contrast to the APX-hardness of the rank-3 scheduling, we show that $R||C_{max}$ is FPT parameterized by p_{max} and d.

▶ **Theorem 2.** *There is an FPT algorithm for $R||C_{max}$ that runs in $2^{2^{O(d\log p_{max})}} + n^{O(1)}$ time.*

Notice that $R||C_{max}$ remains NP-hard even if $p_{max} = 3$ [16] or $d = 1$ [8], therefore parameterizing by only p_{max} or d does not suffice.

We complement this algorithmic result by the following lower bound.

▶ **Theorem 3.** *There is no $2^{2^{o(d\log p_{max})}}$ time algorithm for $R||C_{max}$, unless ETH (Exponential Time Hypothesis) fails.*

The approximability of rank d scheduling is not smooth with respect to the rank d, as is already observed by Bhaskara et al. [2], yet it is FPT parameterized by p_{max} and d, with a running time doubly exponential in d. Furthermore, such a running time is unlikely to be improved significantly, as is suggested by the lower bound.

We also discuss the possibility of replacing the parameter p_{max} by $\bar{p}$, which is the number of distinct processing times in matrix P. It is shown by Goemans and Rothvoss [9] that $P||C_{max}$

is in XP parameterized by $\bar{p}$, i.e., there exists a polynomial time algorithm for $P||C_{max}$ if $\bar{p}$ is a constant. Indeed, they establish a structural theorem on integer programming, through which we can further show that $R||C_{max}$ is in XP parameterized by $\bar{p}$ and d. It remains as an important open problem whether $P||C_{max}$ is FPT parameterized by $\bar{p}$.

▶ **Theorem 4.** *$R||C_{max}$ can be solved in $(\log p_{max})^{2^{O(\zeta)}} + 2^{2^{O(\zeta^2)}}(\log p_{max})^{O(1)}$ time, where $\zeta = 2^{O(d \log \bar{p})}$.*

Related work. Scheduling is a fundamental problem in combinatorial optimization and has received considerable attention in history. In the following we provide a very brief overview with the focus on approximation and parameterized algorithmic results.

In 1988, Hochbaum and Shmoys [11] presented a PTAS for $P||C_{max}$ as well as $Q||C_{max}$. Their algorithm has a running time of $(n/\epsilon)^{O(1/\epsilon^2)}$. Subsequent improvements on the running time of the PTAS can be found in [1, 13]. So far, the best PTAS for $Q||C_{max}$ is due to Jansen, Klein and Verschae [14] and has a running time of $2^{O(1/\epsilon \log^{O(1)} 1/\epsilon)} + O(n)$. It is further shown by Chen, Jansen and Zhang [5] that such a running time is essentially the best possible unless ETH fails, even for $P||C_{max}$. For the unrelated machine scheduling problem $R||C_{max}$, Lenstra, Shmoys and Tardos [16] showed that it does not admit any approximation algorithm with a rato strictly smaller than 1.5 unless $P = NP$. They also provided a 2-approximation algorithm, which was slightly improved to a $(2 - 1/m)$-approximation algorithm by Shchepin et al. [20].

A lot of intermediate models between $R||C_{max}$ and $Q||C_{max}$ or $P||C_{max}$ are studied in literature. In this paper, we are most concerned with the rank of the matrix formed by job processing times on machines, i.e., the rank of $P = (p_{ij})_{m \times n}$. Bhaskara et al. [2] initiated the study on approximation algorithms for $R||C_{max}$ with respect to the parameter rank. They showed that rank-2 scheduling admits a QPTAS, while rank-4 scheduling is already APX-hard. Very recently Chen et al. [6] further improves their result by showing that rank-4 scheduling does not admit any approximation algorithm with a ratio that is strictly smaller than 1.5, unless $P = NP$.

This new model of scheduling with a small matrix rank is closely related to the problem of scheduling unrelated machines of few different types, which is another intermediate model that receives much study in literature [4, 7, 19, 15]. In the problem of scheduling unrelated machines of few different types, there are K different types of machines. If two machines i and i' are of the same type, then for every job j it follows that $p_{ij} = p_{i'j}$. Simply speaking, machines could be divided into K disjoint groups such that machines belonging to the same group are identical. It is shown by Bonifaci and Wiese [4] that if K is a constant, then there exists a PTAS. A PTAS of improved running time was recently presented by Gehrke et al. [19]. It is very easy to see that the problem of scheduling unrelated machines of K different types is actually a special case of the scheduling problem with a matrix of rank K.

Compared with the study on approximation algorithms for the scheduling problem, the study on parameterized algorithms is relatively new. Mnich and Wiese [17] were the first to study FPT algorithms for the scheduling problem. They showed that $P||C_{max}$ is FPT parameterized by p_{max}, the largest job processing times. Meanwhile $R||C_{max}$ is FPT parameterized by the number of machines m and the number of distinct job processing times $\bar{p}$. As all job processing times are integers, $\bar{p}$ is upper bounded by p_{max}. Hence, their results also imply that $R||C_{max}$ is FPT parameterized by m and p_{max}. Very recently, Knop and Koutecký [15] considered the problem of scheduling unrelated machines of few different types, and showed that $R||C_{max}$ is FPT parameterized by p_{max} and K, where K is the number

of different types of machines. FPT algorithms for the scheduling problem with different models have also received much study in literature, see, e.g., [3, 22].

It is, however, not clear whether $R||C_{max}$ is FPT parameterized by K and $\bar{p}$. A recent paper by Goemans and Rothvoss [9] showed that $P||C_{max}$ could be solved in $(\log p_{max})^{2^{O(\bar{p})}}$ time. Therefore $P||C_{max}$ is in XP parameterized by $\bar{p}$, i.e., if there is only a constant number of distinct job processing times, then $P||C_{max}$ could be solved in polynomial time. Indeed, the general structural theorem established in their paper further implies that $R||C_{max}$ is in XP parameterized by K and $\bar{p}$.

2 Preliminaries

Let $P = (p_{ij})_{m\times n}$ with $p_{ij} \in \mathbb{N}$ being the processing time of job j on machine i. Let d be the rank of P. By linear algebra, the matrix P can be expressed as $P = MJ$, where M is an $m \times d$ matrix and J is a $d \times n$ matrix. We can interpret each row vector u_i of M as the d-dimensional *speed vector* of machine i, and each column vector v_j^T of J as the d-dimensional *size vector* of job j. The processing time of job j on machine i is then the product of the two corresponding vectors, i.e., $p_{ij} = u_i \cdot v_j^T$. Bhaskara et al. [2] formally define the scheduling problem with low rank processing time matrix by explicitly giving the speed vector of every machine and the size vector of every job. In our paper, we do not necessarily require that the speed and size vectors are given. If these vectors are not given, we take an arbitrary decomposition of the matrix P into $P = MJ$. Therefore, throughout this paper, we do not necessarily require an entry in a speed vector or a size vector to be an integer or a non-negative number.

Some lower bounds on the running time of algorithms in this paper are based on the following Exponential Time Hypothesis (ETH), which was introduced by Impagliazzo, Paturi, and Zane [12]:

Exponential Time Hypothesis (ETH): There is a positive real δ such that 3SAT with n variables and m clauses cannot be solved in time $2^{\delta n}(n+m)^{O(1)}$.

Using the Sparsification Lemma by Impagliazzo et al. [12], ETH implies that there is no algorithm for 3SAT with n variables and m clauses that runs in time $2^{\delta m}(n+m)^{O(1)}$ for some $\delta > 0$ as well.

3 APX-hardness for rank-3 scheduling

The whole section is devoted to the proof of Theorem 1. For ease of presentation, when we prove the APX-hardness for rank-3 scheduling, we may construct jobs of fractional processing times. However, by scaling we can easily make all the fractional values into integers.

We start with the one-in-three 3SAT problem, which is a variation of the 3SAT problem. An input of the one-in-three 3SAT problem is a boolean formula that is a conjuction of clauses, where each clause is a disjunction of exactly three 3 literals. The formula is satisfied if and only if there exists a truth assignment of variables such that in every clause there is exactly one true literal, i.e., every clause is satisfied by exactly one variable. It is proved in [18] that it is NP-complete to determine whether an arbitrary given instance of the one-in-three 3SAT problem is satisfiable.

We reduce from a variation of the one-in-three 3SAT problem. Given an instance of the one-in-three 3SAT problem, say, I_{sat}, we can apply Tovey's method [21] to transform it into I'_{sat} such that:

- each clause of I_{sat} contains two or three literals;
- each variable appears three times in clauses. Among its three occurrences there are either two positive literals and one negative literal, or one positive literal and two negative literals;
- there exists a truth assignment for I'_{sat} where every clause is satisfied by exactly one literal if and only if there is a truth assignment for I_{sat} where every clause is satisfied by exactly one literal.

The transformation is straightforward. For any variable z, if it only appears once in the clauses, then we add a dummy clause as $(z \vee \neg z)$. Otherwise suppose it appears $d \geq 2$ times in the clauses, then we replace its d occurrences with d new variables as z_1, z_2, $\cdots$, z_d, and meanwhile add d clauses as $(z_1 \vee \neg z_2)$, $(z_2 \vee \neg z_3)$, $\cdots$, $(z_d \vee \neg z_1)$ to enforce that these new variables should take the same truth assignment. It is not difficult to verify that the constructed instance satisfies the above requirements.

Throughout the following part of this section we assume that I'_{sat} contains n variables and m clauses. Let ϵ be an arbitrary small positive number. Let $\tau = 2^3$, $r = 2^{11}\tau = 2^{14}$, $N = n/\epsilon^2$. We will construct an instance I_{sch} of the rank-3 scheduling problem such that:

- if there is a truth assignment for I'_{sat} where every clause is satisfied by exactly one variable, then I_{sch} admits a feasible schedule whose makespan is $r + c\epsilon$ for some constant c;
- if I_{sch} admits a feasible schedule whose makespan is strictly less than $r + 1$, then there exists a truth assignment for I'_{sat} where every clause is satisfied by exactly one variable.

We claim that, given the above construction, Theorem 1 follows. To see why, suppose on the contrary that there exists a $(1 + \rho)$-approximation algorithm for some constant $\rho < 2^{-14}$. We set $\epsilon = \frac{1-r\rho}{c\rho} = \frac{1-2^{14}\rho}{c\rho}$, and apply this algorithm to the constructed instance I_{sch}. There are two possibilities. If I'_{sat} is satisfiable, then the approximation algorithm returns a feasible solution whose makespan is at most $(r + c\epsilon)(1 + \rho) = r + r\rho + c\rho \cdot \epsilon < r + 1$. If I'_{sat} is not satisfiable, then I_{sch} does not admit a feasible schedule whose makespan is strictly less than $r + 1$, i.e., any feasible schedule has a makespan at least $r + 1$, whereas the $(1 + \rho)$-approximation algorithm returns a solution whose makespan is at least $r + 1$. Thus, we can use the $(1 + \rho)$-approximation algorithm to determine the satisfiability of I'_{sat}, and consequently the satisfiability of I_{sat} in polynomial time, which contradicts the NP-hardness of the one-in-three 3SAT problem.

Construction of the scheduling instance. To construct the scheduling instance, we construct the size vector of every job and speed vector of every machine. Each vector is a triple of three positive numbers. The processing time of a job on a machine is then the inner product of the two corresponding vectors. As we describe in Section 2, the constructed instance is a feasible instance of rank-3 scheduling.

Recall that $r = 2^{14}$, $\tau = 2^3$, $N = n/\epsilon^2$. Indeed, if we do not care much about the value of ρ and only want to show APX-hardness, it suffices to think r as some value significantly larger than τ. For a job j we denote by $s(j)$ its size vector.

We construct two main kinds of jobs, element jobs and tuple jobs. In the following we first construct element jobs, which are further divided into variable jobs, truth-assignment jobs, clause jobs and dummy jobs.

- **Variable jobs.** For every variable z_i, we construct 8 variable jobs, $v^{\gamma}_{i,k}$ for $k = 1, 2, 3, 4$ and $\gamma = T, F$. Their size vectors are:

$$s(v^T_{i,1}) = (\epsilon N^{4i+1}, 0, r/8 - 10\tau - 2), \qquad s(v^T_{i,2}) = (\epsilon N^{4i+2}, 0, r/8 - 20\tau - 2),$$

$$s(v_{i,3}^T) = (\epsilon N^{4i+3}, 0, r/8 - 18\tau - 2), \qquad s(v_{i,4}^T) = (\epsilon N^{4i+4}, 0, r/8 - 12\tau - 2).$$
$$s(v_{i,k}^F) = s(v_{i,k}^T) - (0, 0, 2), k = 1, 2, 3, 4$$

- **Truth-assignment jobs.** For every variable z_i, we construct eight truth-assignment jobs, a_i^γ, b_i^γ, c_i^γ d_i^γ with $\gamma = T, F$. Their size vectors are:

$$s(a_i^T) = (0, \epsilon N^i, r/64 + 2\tau + 1), \qquad s(b_i^T) = (0, \epsilon N^i, r/64 + 4\tau + 1),$$
$$s(c_i^T) = (0, \epsilon N^i, r/64 + 8\tau + 1), \qquad s(d_i^T) = (0, \epsilon N^i, r/64 + 16\tau + 1).$$
$$s(\tau_i^F) = s(\tau_i^T) + (0, 0, 1), \tau = a, b, c, d$$

- **Clause jobs.** For every clause e_j, if it contains two literals, then we construct two clause jobs, u_j^T and u_j^F. Otherwise it contains three literals, and we construct three clause jobs, one u_j^T and two u_j^F. Their size vectors are:

$$s(u_j^T) = (0, \epsilon N^{N+j}, r/4 + 2), \qquad s(u_j^F) = (0, \epsilon N^{N+j}, r/4 + 4).$$

- **Dummy jobs.** We construct $2n - m$ true dummy jobs ϕ^T and $m - n$ false dummy jobs ϕ^F. Their size vectors are:

$$s(\phi^F) = (0, 0, r/16 + 4), \qquad s(\phi^T) = (0, 0, r/16 + 2).$$

We finish the description of the element jobs and now define tuple jobs. Indeed, there is a one-to-one correspondence between tuple jobs and machines. For ease of description, we first construct machines, and then construct tuple jobs.

We construct $8n$ machines, which are further divided into truth-assignment machines, clause machines and dummy machines. For a machine i we denote by $g(i)$ its speed vector.

- **Truth-assignment machines.** For every variable z_i, we construct $4n$ truth-assignment machines, denoted as $(v_{i,1}, a_i, c_i)$, $(v_{i,2}, b_i, d_i)$, $(v_{i,3}, a_i, d_i)$, $(v_{i,4}, b_i, c_i)$. The symbol of a machine actually indicates the jobs that we will put on it. The speed vectors are:

$$g(v_{i,1}, a_i, c_i) = (N^{-4i-1}, N^{-i}, 1), \qquad g(v_{i,2}, b_i, d_i) = (N^{-4i-2}, N^{-i}, 1),$$
$$g(v_{i,3}, a_i, d_i) = (N^{-4i-3}, N^{-i}, 1), \qquad g(v_{i,4}, b_i, c_i) = (N^{-4i-4}, N^{-i}, 1).$$

- **Clause machines.** For every clause e_j: if the positive (or negative) literal z_i (or $\neg z_i$) appears in it for the first time (i.e., it does not appear in e_k for $k < j$), then we construct a clause machine $(v_{i,1}, u_j)$ (or $(v_{i,3}, u_j)$); if it appears for the second time, then we construct a clause machine $(v_{i,2}, u_j)$ (or $(v_{i,4}, u_j)$). The speed vectors are:

$$g(v_{i,k}, u_j) = (N^{-4i-k}, N^{-N-j}, 1).$$

- **Dummy machines.** Recall that for every variable, in all the clauses there are either one positive literal and two negative literals, or two positive literals and one negative literal. If z_i appears once and $\neg z_i$ appears twice, then we construct a dummy machine $(v_{i,2}, \phi)$, otherwise we construct a dummy machine $(v_{i,4}, \phi)$. The speed vectors are:

$$g(v_{i,2}, \phi) = (N^{-4i-2}, 0, 1), \qquad g(v_{i,4}, \phi) = (N^{-4i-4}, 0, 1).$$

According to our construction, it is not difficult to verify that if z_i appears once and $\neg z_i$ appears twice, then we construct machines $(v_{i,k}, u_{j_k})$ for $k = 1, 3, 4$, $1 \le j_k \le m$, and machine $(v_{i,2}, \phi)$. Otherwise we construct machines $(v_{i,k}, u_{j_k})$ for $k = 1, 2, 3$, $1 \le j_k \le m$, and machine $(v_{i,4}, \phi)$. This completes the construction of machines.

- **Tuple jobs.** Finally, we construct tuple jobs. There is one tuple job corresponding to each machine. For simplicity, tuple jobs corresponding to truth-assignment, clause, dummy machines are called tuple-truth-assignment, tuple-clause, tuple-dummy jobs, respectively. We also use the symbol of a machine to denote its corresponding tuple job. The size vectors of tuple jobs are:

$$s(v_{i,1}, a_i, c_i) = (\epsilon N^{4i+1}, \epsilon N^i, 27r/32), \qquad s(v_{i,2}, b_i, d_i) = (\epsilon N^{4i+2}, \epsilon N^i, 27r/32),$$
$$s(v_{i,3}, a_i, d_i) = (\epsilon N^{4i+3}, \epsilon N^i, 27r/32), \qquad s(v_{i,4}, b_i, c_i) = (\epsilon N^{4i+4}, \epsilon N^i, 27r/32).$$
$$s(v_{i,1}, u_j) = (0, \epsilon N^{N+j}, 5r/8 + 10\tau), \qquad s(v_{i,2}, u_j) = (0, \epsilon N^{N+j}, 5r/8 + 20\tau),$$
$$s(v_{i,3}, u_j) = (0, \epsilon N^{N+j}, 5r/8 + 18\tau), \qquad s(v_{i,4}, u_j) = (0, \epsilon N^{N+j}, 5r/8 + 12\tau).$$
$$s(v_{i,2}, \phi) = (0, N^{2N}, 13r/16 + 20\tau), \qquad s(v_{i,4}, \phi) = (0, N^{2N}, 13r/16 + 12\tau).$$

Note that the size vectors of tuple-dummy jobs and tuple-clause jobs are actually independent of the index i.

This completes the construction of the whole scheduling instance. Recall that the processing time of a job on a machine is the inner product of the two corresponding vectors. Given our construction of machines and jobs, we have the following simple observation.

▶ **Observation 5.** *Let x be an arbitrary job whose size vector is $s(x) = (s_1(x), s_2(x), s_3(x))$. Then the processing time of x is at least $s_3(x)$ on every machine. Furthermore, its processing time is $s_3(x) + O(\epsilon)$ if one of the following holds:*

- *x is an element job and is scheduled on a machine whose symbol contains x;*
- *x is a tuple job and is scheduled on its corresponding machine.*

We remark that, it is possible for a job x to have a processing time $s_3(x) + O(\epsilon)$ on a machine even if the two conditions of the above observation do not hold, that is, the two conditions are not necessary.

The overall structure of our construction is similar to that of the paper [5] by Chen, Jansen and Zhang. We construct variable jobs corresponding to variables, clause jobs corresponding to clauses, truth-assignment jobs corresponding to the truth assignment of the SAT instance. Such kinds of jobs also appear in the reduction of [5] when they reduce 3SAT to the scheduling problem $P||C_{max}$. However, the reduction of Chen et al. [5] is for $P||C_{max}$ which belongs to the rank 1 scheduling problem and does not work for higher ranks. To show APX-hardness, we need to construct completely different job processing times.

We first prove the following lemma.

▶ **Lemma 6.** *If there exists a truth assignment for I'_{sat} where every clause is satisfied by exactly one variable, then I_{sch} admits a feasible schedule whose makespan is $r + O(\epsilon)$.*

We give a brief overview of the proof and the reader may refer to the full version of this paper for details. It can be found at `https://www.researchgate.net/publication/313852592_Parameterized_and_approximation_results_for_scheduling_with_a_low_rank_processing_time_matrix`. We schedule jobs according to the first two columns of Table 1. Notice that the first two columns specify which job is on which machine, except that for an element job, say, a_i, it does not specify whether it is a_i^T or a_i^F. There are two possibilities regarding to the superscripts of element jobs on every machine, as is indicated by the third and fourth columns of Table 1. Either way ensures that the total processing times of jobs on each machine is $r + O(\epsilon)$. The technical part of the proof shows how to choose a proper way for every machine (based on the truth assignment of I'_{sat}) so that all the jobs get scheduled.

Table 1 Overview of the schedule.

machines	jobs	Feasible ways of scheduling	
$(v_{i,1}, a_i, c_i)$	$v_{i,1}$, a_i, c_i, $(v_{i,1}, a_i, c_i)$	$v_{i,1}^T$, a_i^T, c_i^T	$v_{i,1}^F$, a_i^F, c_i^F
$(v_{i,2}, b_i, d_i)$	$v_{i,2}$, b_i, d_i, $(v_{i,2}, b_i, d_i)$	$v_{i,2}^T$, b_i^T, d_i^T	$v_{i,2}^F$, b_i^F, d_i^F
$(v_{i,3}, a_i, d_i)$	$v_{i,3}$, a_i, d_i, $(v_{i,3}, a_i, d_i)$	$v_{i,3}^T$, a_i^T, d_i^T	$v_{i,3}^F$, a_i^F, d_i^F
$(v_{i,4}, b_i, c_i)$	$v_{i,4}$, b_i, c_i, $(v_{i,4}, b_i, c_i)$	$v_{i,4}^T$, b_i^T, c_i^T	$v_{i,4}^F$, b_i^F, c_i^F
$(v_{i,k}, u_j)$	$v_{i,k}$, u_j, $(v_{i,k}, u_j)$	$v_{i,k}^T$, u_j^T	$v_{i,k}^F$, u_j^F
$(v_{i,k}, \phi)$	$v_{i,k}$, ϕ, $(v_{i,k}, \phi)$	$v_{i,k}^T$, ϕ^T	$v_{i,k}^F$, ϕ^F

▶ **Lemma 7.** *If there is a solution for I_{sch} whose makespan is strictly less than $r+1$, then there exists a truth assignment for I'_{sat} where every clause is satisfied by exactly one literal.*

According to Observation 5, the total processing time of all jobs in any feasible solution is at least the summation of the third coordinates of all jobs, which is at least $8nr$ with simple calculations. Let Sol^* be the solution whose makespan is strictly less than $r+1$. We have the following structural lemma.

▶ **Lemma 8.** *In Sol^*, the followings are true:*

- *on a truth-assignment machine, there is exactly one tuple-truth-assignment job, two truth-assignment jobs and one variable job;*
- *on a clause machine, there is exactly one tuple-clause job, one clause job and one variable job;*
- *on a dummy machine, there is exactly one tuple-dummy job, one dummy job and one variable job.*

Proof Idea. The first and second coordinates of the speed and size vectors restrict the scheduling of jobs, e.g., by checking the second coordinate we can conclude that the processing time of a tuple-dummy job is $\Omega(N)$ on any clause machine or truth-assignment machine, hence it has to be on a dummy machine. The third coordinate of a size vector gives a lower bound on the job processing time and allows us to derive some overall structure, e.g., each tuple job has a processing time at least $5r/8$, hence there can not be two tuple jobs on one machine. Given that the number of tuple jobs equals the number of machines, there is exactly one tuple job on one machine. Lemma 8 follows by combining the above basic idea with a careful analysis of job processing times. The reader may refer to the full version of this paper for all the details. ◀

A machine is called matched, if all the jobs on this machine coincide with the symbol of this machine, i.e., jobs are scheduled according to the second column of Table 1. Specifically, we say a machine is matched with respect to variable, or clause, or truth-assignment, or tuple jobs, if the variable, or clause, or truth-assignment, or tuple jobs on this machine coincide with the symbol of this machine.

▶ **Lemma 9.** *We may assume that every machine is matched with respect to variable jobs.*

Proof. Consider the eight jobs $v_{n,k}^\gamma$ where $\gamma = T, F$, $k = 1, 2, 3, 4$. For any machine denoted as $(v_{j,k}, *)$ or $(v_{j,k}, *, *)$, the first coordinate of its speed vector is N^{-4j-k}, thus the processing time of $v_{n,k}$ on this machine becomes $\Omega(\epsilon N)$ if $j < n$. Furthermore, $v_{n,4}$ can only be on machines whose symbols are $(v_{n,4}, *)$ or $(v_{n,4}, *, *)$, since if it is put on a machine whose symbol is $(v_{n,k}, *)$ or $(v_{n,k}, *, *)$ where $k \in \{1, 2, 3\}$, then its processing time also becomes $\Omega(\epsilon N)$. Notice that there are two jobs with the symbol $v_{n,4}$ (one true job $v_{n,4}^T$ and one

false job $v^F_{n,4}$), and two machines with the symbol $(v_{n,4}, *)$ or $(v_{n,4}, *, *)$ (either machines $(v_{n,4}, b_n, c_n)$ and $(v_{n,4}, \phi)$, or machines $(v_{n,4}, b_n, c_n)$ and $(v_{i,4}, u_{j_n})$ for some j_n). According to Lemma 8, there is one variable job on every machine. Thus the two machines with the symbol $(v_{n,4}, *)$ or $(v_{n,4}, *, *)$ are matched with respect to variable jobs.

Next we consider the two variable jobs $v_{n,3}$. Using the same arguments as above, we can show that they can only be scheduled on a machine whose symbol is $(v_{n,k}, *)$ or $(v_{n,k}, *, *)$ where $k \in \{3, 4\}$. Furthermore, we have already shown that the variable job on a machine with the symbol $(v_{n,4}, *)$ or $(v_{n,4}, *, *)$ is $v_{n,4}$, and by Lemma 8 there can only be one variable job on every machine. Hence, the two jobs $v_{n,3}$ can only be on the two machines whose symbols are $(v_{n,3}, *)$ or $(v_{n,3}, *, *)$, and consequently these two machines are matched with respect to variable jobs.

Iteratively applying the above arguments we can prove that every machine is matched with respect to variable jobs. ◀

We can further prove that every machine is matched with respect to clause jobs, tuple jobs and truth-assignment jobs, and therefore the following Lemma 10 is proved. The basic idea is similar to the proof of Lemma 9, but a more careful estimation of job processing times is required. A case by case analysis is needed several times to eliminate certain ways of scheduling. The reader may refer to the full version of this paper for details.

▶ **Lemma 10.** *We may assume that every machine is matched in Sol^*.*

Finally, we consider the superscripts of jobs on every machine. A machine is called truth benevolent, if except the tuple job, all the jobs on it are either all true or all false, i.e., jobs are scheduled according to the third or fourth column of Table 1. The following lemma follows by a case by case analysis showing that other ways of scheduling will lead to a total processing time larger than $r + 1$ on some machine.

▶ **Lemma 11.** *Every machine is truth benevolent.*

Proof of Lemma 7. According to Lemma 11, for every $1 \le i \le n$, on truth-assignment machines jobs are either scheduled as $(v^T_{i,1}, a^T_i, c^T_i)$, $(v^T_{i,2}, b^T_i, d^T_i)$, $(v^F_{i,3}, a^F_i, d^F_i)$, $(v^F_{i,4}, a^F_i, c^F_i)$ or $(v^F_{i,1}, a^F_i, c^F_i)$, $(v^F_{i,2}, b^F_i, d^F_i)$, $(v^T_{i,3}, a^T_i, d^T_i)$, $(v^T_{i,4}, a^T_i, c^T_i)$. If the former case happens, we let the variable z_i be false, otherwise we let z_i be true. We prove that, by assigning the truth value in this way, every clause of I'_{sat} is satisfied by exactly one literal.

Consider any clause, say, e_j. It contains two or three variables and we let them be v_{i_1,k_1}, v_{i_2,k_2} and v_{i_3,k_3} where $k_1, k_2, k_3 \in \{1, 2, 3, 4\}$ (if it contains two variables then v_{i_3,k_3} does not exist). Since there is one u^T_j and one or two u^F_j, we assume that u^T_j is scheduled with $v^T_{i_1,k_1}$.

We prove that e_j is satisfied by variable z_{i_1}. Notice that according to Lemma 10 and Lemma 11, u^T_j and $v^T_{i_1,k_1}$ are scheduled together on machine (v_{i_1,k_1}, u_j). There are two possibilities. Suppose $k_1 \in \{1, 2\}$. According to the construction of the scheduling instance, machine (v_{i_1,k_1}, u_j) is constucted if the positive literal z_i appears in clause e_j for the first or second time. According to our truth assignment in the paragraph above, variable z_i is true, for otherwise $v^T_{i_1,k_1}$ is scheduled with a^T_i, c^T_i or b^T_i, d^T_i, thus e_j is satisfied by variable z_{i_1}. Otherwise $k_1 \in \{3, 4\}$. According to the construction of the scheduling instance, machine (v_{i_1,k_1}, u_j) is constructed if the negative literal $\neg z_i$ appears in clause e_j for the first or second time. Again according to our truth assignment in the paragraph above, the variable z_i is false, thus e_j is satisfied by variable z_{i_1}.

We prove that e_j is *not* satisfied by variable z_{i_2} or z_{i_3}. Consider z_{i_2}. Notice that according to Lemma 10 and Lemma 11, u^F_j and $v^F_{i_2,k_2}$ are scheduled together on machine (v_{i_2,k_2}, u_j). There are two possibilities. Suppose $k_2 \in \{1, 2\}$. According to the construction of the

scheduling instance, machine (v_{i_2,k_2}, u_j) is constructed if the positive literal z_{i_2} appears in e_j for the first or second time. Meanwhile, variable z_{i_2} is false because otherwise $v^F_{i_2,k_2}$ is scheduled with a^F_i, c^F_i or b^F_i, d^F_i according to our truth assignment of variables. Thus e_j is not satisfied by variable z_{i_2}. Similarly, we can prove that if $k_2 \in \{3, 4\}$, e_j is not satisfied by variable z_{i_2}, either. The proof is the same for variable z_{i_3}, if it exists. ◀

4 Parameterized algorithms and lower bounds

4.1 Parameterizing by by p_{max} and d

We show $R||C_{max}$ is FPT parameterized by p_{max} and the rank d. It is indeed a combination of a simple observation together with the following result by Knop and Koutecký [15].

▶ **Theorem 12** ([15]). *$R||C_{max}$ is FPT parameterized by p_{max} and K, where K is the number of different kinds of machines.*

▶ Remark. In [15], machine kind is such defined that if two machines are of the same kind, then the processing time of every job is the same on them. Using our terminology, K is the number of distinct speed vectors. It is implicitly shown in [15] that the FPT algorithm runs in $2^{O(\Theta^2 K \log p_{max})} + n^{O(1)}$ time, where Θ is the number of distinct size vectors.

We observe that, if both the numbers of distinct speed vectors and size vectors are bounded by some function of p_{max} and d, then Theorem 2 follows directly from Theorem 12. In the following we show an even stronger result.

▶ **Lemma 13.** *Let $\bar{p}$ be the number of distinct processing times in the matrix $P = (p_{ij})_{m\times n}$, and d be the rank of this matrix. There are at most $\bar{p}^d + 1$ distinct speed vectors, and $\bar{p}^d + 1$ distinct size vectors.*

Proof. We show that the number of distinct speed vectors is bounded by $\bar{p}^d + 1$. Due to symmetry the number of distinct size vectors is also bounded by the same value.

Consider all the size vectors. Since the matrix P has rank d, we are able to find d distinct size vectors that are linearly independent. Let them be $v_1, v_2, \cdots, v_d$. Suppose there are $n' \geq \bar{p}^d + 1$ distinct speed vectors and we consider each $u_i \cdot v_1^T$ (recall that u_i is the speed vector of machine i). As jobs have at most $\bar{p}$ distinct processing times, the product $u_i \cdot v_1^T$ can take at most $\bar{p}$ distinct values. According to the pigeonhole principle there exist at least $\lceil n'/\bar{p} \rceil \geq \bar{p}^{d-1} + 1$ distinct speed vectors leading to the same product. Similarly, among these speed vectors we can further select at least $\bar{p}^{d-2} + 1$ ones such that their product with v_2 are the same. Carry on the argument, eventually we can find at least 2 distinct speed vectors, say, u_1 and u_2, such that their product with $v_1, v_2, \cdots, v_d$ are always the same, i.e., $(u_1 - u_2) \cdot v_i^T = 0$ for $1 \leq i \leq d$. However, $v_1, v_2, \cdots, v_d$ are linearly independent, hence $u_1 - u_2 = 0$, which contradicts the fact that u_1 and u_2 are different. ◀

Next we prove Theorem 3, which suggests that the FPT algorithm in Theorem 2 is essentially the best possible under ETH. We reduce from 3-dimensional matching.

3-Dimensional Matching (3DM)
Input: 3 disjoint sets of elements $W = \{w_1, w_2, \cdots, w_n\}$, $X = \{x_1, x_2, \cdots, x_n\}$, $Y = \{y_1, y_2, \cdots, y_n\}$ such that $|W| = |X| = |Y| = n$. A set $T \subseteq W \times X \times Y$.
Output: Decide whether there exists a perfect matching of size n, i.e., a subset $T' \subseteq T$ such that $|T| = n$, and for any two distinct triples $(w, x, y), (w', x', y')$ it follows that $w \neq w'$, $x \neq x'$, $y \neq y'$.

The traditional NP-hardness proof (see, e.g., [8]) for the 3-dimensional matching problem reduces a $3SAT$ instance of n variables to a $3DM$ instance with $O(n)$ elements, hence the following corollary follows.

▶ **Corollary 14.** *Assuming ETH, there is no $2^{o(n)}$ time algorithm for $3DM$.*

Given an arbitrary instance of $3DM$, we construct in the following a scheduling instance with $|T|$ machines and $3|T|$ jobs such that the scheduling instance admits a feasible schedule of makespan at most 11109Γ if and only if the $3DM$ instance admits a perfect matching, where $\Gamma = \sum_{i=1}^{\tau} i \cdot (\tau - i)$ with integer τ being the smallest integer such that $\tau! \geq n$ (consequently, $\tau = O(\log n / \log\log n)$). Furthermore, the scheduling instance we construct satisfies that $d = O(\tau)$, $p_{max} = \tau^{O(1)}$. Now it is easy to verify that $d \log p_{max} = O(\log n)$. We claim that Theorem 3 follows from the reduction above. To see why, suppose on the contrary that Theorem 3 is false. Then there exists an algorithm of running time $2^{2^{o(d \log p_{max})}}$ for $R||C_{max}$. We apply this algorithm to the constructed scheduling instance. As $d \log p_{max} = O(\log n)$, in $2^{o(n)}$ time the algorithm determines whether the constructed scheduling instance admits a feasible schedule of makespan at most 11109Γ, and consequently whether the given $3DM$ instance admits a perfect matching. This, however, is a contradiction to Corollary 14.

Construction of the scheduling instance. Note that $\tau! \geq n$, hence we can map each integer $1 \leq i \leq n$ to a unique permutation of integers $\{1, 2, \cdots, \tau\}$. Let σ be such a mapping. For ease of notation, we denote by σ_i the permutation that i is mapped to by σ. Consequently $\sigma_i(k)$ denotes the integer on the k-th position of the permutation σ_i.

We construct $|T|$ machines, each corresponding to one triple $(w_i, x_j, y_k) \in T$. The machine corresponding to (w_i, x_j, y_k) has the speed vector $(1, \phi(w_i), \phi(x_j), \phi(y_k))$ where

$$\phi(w_i) = (\sigma_i(1), \sigma_i(2), \cdots, \sigma_i(\tau)), \quad \phi(x_j) = (\sigma_j(1), \sigma_j(2), \cdots, \sigma_j(\tau)),$$
$$\phi(y_k) = (\sigma_k(1), \sigma_k(2), \cdots, \sigma_k(\tau)).$$

For every element $z \in W \cup X \cup Y$, let $\eta(z)$ denote the number of occurrences of z in the set of triples T. We construct $\eta(z)$ jobs for every element z. Among the $\eta(z)$ jobs, there is one true job of size vector $(g_T(z) \cdot \Gamma, \psi_w(z), \psi_x(z), \psi_y(z))$. Each of the remaining $\eta(z) - 1$ jobs is called a false job, having a size vector of $(g_F(z) \cdot \Gamma, \psi_w(z), \psi_x(z), \psi_y(z))$, where

$$\psi_w(w_i) = (\tau - \sigma_i(1), \tau - \sigma_i(2), \cdots, \tau - \sigma_i(\tau)), \psi_w(x_j) = \psi_w(y_k) = \underbrace{(0, 0, \cdots, 0)}_{\tau},$$
$$\psi_x(x_j) = (\tau - \sigma_j(1), \tau - \sigma_j(2), \cdots, \tau - \sigma_j(\tau)), \psi_x(w_i) = \psi_x(y_k) = \underbrace{(0, 0, \cdots, 0)}_{\tau},$$
$$\psi_y(y_k) = (\tau - \sigma_k(1), \tau - \sigma_k(2), \cdots, \tau - \sigma_k(\tau)), \psi_y(w_i) = \psi_y(x_j) = \underbrace{(0, 0, \cdots, 0)}_{\tau},$$

$$g_T(w_i) = 10^2 + 4, \quad g_T(x_j) = 10^3 + 1, \quad g_T(y_k) = 10^4 + 1,$$
$$g_F(w_i) = 10^2 + 2, \quad g_F(x_j) = 10^3 + 2, \quad g_F(y_k) = 10^4 + 2.$$

We show that the constructed scheduling instance admits a feasible solution of makespan at most 11109Γ if and only if the $3DM$ instance admits a perfect matching.

Suppose the given $3DM$ instance admits a perfect matching T'. For every $(w_i, x_j, y_k) \in T'$, we put the three true jobs corresponding to w_i, x_j, y_k onto the machine corresponding to this triple. It is easy to verify that the total processing time of the three jobs sum to exactly 11109Γ. For every $(w_{i'}, x_{j'}, y_{k'}) \in T \setminus T'$, we put three false jobs corresponding to $w_{i'}$, $x_{j'}$, $y_{k'}$ onto the machine corresponding to this triple. It is also easy to verify that the total

processing times sum up to 11109Γ. Note that there is one true job corresponding to each element, while every element appears once in T', all the jobs are scheduled and we derive a feasible schedule of makespan 11109Γ.

Suppose the scheduling instance admits a feasible schedule of makespan bounded by 11109Γ, we prove in the following that the $3DM$ instance admits a perfect matching.

Consider the processing time of a job corresponding to z on a machine corresponding to (w_i, x_j, y_k). The processing time is $g_T(z) \cdot \Gamma + \lambda(z, (w_i, x_j, y_k))$, if it is a true job, or $g_F(z) \cdot \Gamma + \lambda(z, (w_i, x_j, y_k))$ otherwise. We observe that the processing time consists of two parts. The *machine-independent value*, which is $g_T(z) \cdot \Gamma$ or $g_F(z) \cdot \Gamma$ that only relies on the job, and the *machine-dependent value*, which is $\lambda(z, (w_i, x_j, y_k))$. The following lemma provides a lower bound on $\lambda(z, (w_i, x_j, y_k))$.

▶ **Lemma 15.** *For any element z and triple (w_i, x_j, y_k), the following is true.*

$$\lambda(z, (w_i, x_j, y_k)) = (1, \phi(w_i), \phi(x_j), \phi(y_k)) \cdot (0, \psi_w(z), \psi_x(z), \psi_y(z))^T \geq \Gamma.$$

Furthermore, the equality holds if and only if $z = w_i$ or $z = x_j$ or $z = y_k$.

Lemma 15 follows immediately from the following *Rearrangment Inequality* [10].

▶ **Theorem 16** (Rearrangment Inequality)**.** *Let $a_1 < a_2 < \cdots < a_n$, $b_1 < b_2 < \cdots < b_n$ be two lists of real numbers, then*

$$a_n b_1 + a_{n-1} b_2 + \cdots + a_1 b_n \leq a_{\pi(1)} b_1 + a_{\pi(2)} b_2 + \cdots + a_{\pi(n)} b_n \leq a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

holds for any permutation π. Furthermore, the lower bound is attained if and only if $\pi(i) = n + 1 - i$, and the upper bound is attained if and only if $\pi(i) = i$.

▶ **Lemma 17.** *A job corresponding to an element z is scheduled on a machine corresponding to a triple that contains z.*

Proof. We sum up the processing time of all jobs. There are n true jobs and $|T| - n$ false jobs corresponding to elements of W. The machine-independent value of these jobs sum up to $104n\Gamma + 102(|T| - n)\Gamma = 102|T| \cdot \Gamma + 2n\Gamma$. Similarly, it is easy to verify that the machine-independent value of jobs corresponding to elements of X and Y sum up to $1001n\Gamma + 1002(|T| - n)\Gamma = 1002|T| \cdot \Gamma - n\Gamma$ and $10001n\Gamma + 10002(|T| - n)\Gamma = 10002|T| \cdot \Gamma - n\Gamma$, respectively. Hence the machine-independent value of all jobs sum up to $11106|T| \cdot \Gamma$. As the makespan is 11109Γ, the total processing time of all jobs is at most $11109|T| \cdot \Gamma$, implying that the summation of machine-dependent value of all jobs is at most $3|T| \cdot \Gamma$. According to Lemma 15, the machine-dependent value of each job is at least Γ, regardless of which machine it is scheduled on. Given that there are $3|T|$ jobs, the machine-dependent value of every job is exactly Γ. Again due to Lemma 15, a job corresponding to z must be scheduled on machine corresponding to a triple that contains z. ◀

For simplicity, we call a job corresponding to an element of W (X or Y) as a w-job (x-job or y-job). We have the following lemma.

▶ **Lemma 18.** *There are three jobs on each machine, one w-job, one x-job and one y-job.*

Proof. Notice that the machine-dependent value of each job in the schedule is exactly Γ, hence the machine-independent value of jobs on each machine sum up to at most 11106Γ. Notice that the machine-independent value of a y-job at least $10^4\Gamma$, there is at most one y-job on each machine. Furthermore, there are exactly $|T|$ machines and y-jobs, hence, there is exactly one y-job on each machine. Similarly, we can show that there is one x-job and one w-job on each machine. ◀

Combining the above two lemmas, we have the following.

▶ **Lemma 19.** *On the machine corresponding to* (w_i, x_j, y_k)*, the three jobs correspond to* w_i*,* x_j*,* y_k*, respectively.*

Finally, we check whether jobs are true or false on each machine. Indeed, as the machine-independent value of the three jobs on each machine sum up to 11106Γ, they are either all true jobs or all false jobs, hence, there are n machines on which all jobs are true jobs, and the triples corresponding to these machine form a perfect matching.

4.2 Parameterizing by $\bar{p}$ and d

We remark that, although it is not written explicitly, the general structural theorem in [9] actually implies an XP algorithm for $R||C_{max}$ parameterized by $\bar{p}$ and K, where K is the number of different kinds of machines. Combining this result with Lemma 13, Theorem 4 follows directly. For the completeness of this paper, we give all the proofs in the full version of this paper.

References

1 Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.

2 Aditya Bhaskara, Ravishankar Krishnaswamy, Kunal Talwar, and Udi Wieder. Minimum makespan scheduling with low rank processing times. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 937–947. Society for Industrial and Applied Mathematics, 2013.

3 Hans L. Bodlaender and Michael R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995.

4 Vincenzo Bonifaci and Andreas Wiese. Scheduling unrelated machines of few different types. *arXiv preprint arXiv:1205.0974*, 2012.

5 Lin Chen, Klaus Jansen, and Guochuan Zhang. On optimality of exact and approximation algorithms for scheduling problems. Technical report, Christian-Albrechts-Universität Kiel, 2013. Report No. 1303. URL: `http://www.uni-kiel.de/journals/receive/jportal_jparticle_00000034`.

6 Lin Chen, Deshi Ye, and Guochuan Zhang. An improved lower bound for rank four scheduling. *Operations Research Letters*, 42(5):348–350, 2014.

7 Lin Chen, Deshi Ye, and Guochuan Zhang. Online scheduling of mixed CPU-GPU jobs. *International Journal of Foundations of Computer Science*, 25(06):745–761, 2014.

8 Michael R. Gary and David S. Johnson. Computers and intractability: A guide to the theory of NP-completeness, 1979.

9 Michel X. Goemans and Thomas Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 830–839. Society for Industrial and Applied Mathematics, 2014.

10 Godfrey Harold Hardy, George Polya, and John Edensor Littlewood. *Inequalities*. Cambridge University Press, 1952.

11 Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM journal on computing*, 17(3):539–551, 1988.

12 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 653–662. IEEE, 1998.

13 Klaus Jansen. An EPTAS for scheduling jobs on uniform processors: using an milp relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24(2):457–485, 2010.

14 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *arXiv preprint arXiv:1604.07153*, 2016.

15 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *arXiv preprint arXiv:1603.02611*, 2016.

16 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

17 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1-2):533–562, 2015.

18 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.

19 Jakob Schikowski. A PTAS for scheduling unrelated machines of few different types. In *SOFSEM 2016: Theory and Practice of Computer Science: 42nd International Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 23-28, 2016, Proceedings*, volume 9587, page 290. Springer, 2016.

20 Evgeny V. Shchepin and Nodari Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127–133, 2005.

21 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

22 René van Bevern, Robert Bredereck, Laurent Bulteau, Christian Komusiewicz, Nimrod Talmon, and Gerhard J. Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. *arXiv preprint arXiv:1605.00901*, 2016.