On the Complexity of Scheduling Problems with a Fixed Number of Parallel Identical Machines

Klaus Jansen(✉) and Kai Kahler(✉)

Department of Computer Science, Kiel University, Kiel, Germany
{kj,kka}@informatik.uni-kiel.de

Abstract. In parallel machine scheduling, we are given a set of jobs, together with a number of machines and our goal is to decide for each job, when and on which machine(s) it should be scheduled in order to minimize some objective function. Different machine models, job characteristics and objective functions result in a multitude of scheduling problems and many of them are NP-hard, even for a fixed number of identical machines. In this work, we give conditional running time lower bounds for a large number of scheduling problems, indicating the optimality of some classical algorithms. Most notably, we show that the algorithm by Lawler and Moore for $1||\sum w_jU_j$ and $Pm||C_{\max}$, as well as the algorithm by Lee and Uzsoy for $P2||\sum w_jC_j$ are probably optimal. There is still small room for improvement for the $1|Rej \leq Q|\sum w_jU_j$ algorithm by Zhang et al., the algorithm for $1||\sum T_j$ by Lawler and the FPTAS for $1||\sum w_jU_j$ by Gens and Levner. We also give a lower bound for $P2|any|C_{\max}$ and improve the dynamic program by Du and Leung from $\mathcal{O}(nP^2)$ to $\mathcal{O}(nP)$, matching this new lower bound. Here, P is the sum of all processing times. The same idea also improves the algorithm for $P3|any|C_{\max}$ by Du and Leung from $\mathcal{O}(nP^5)$ to $\mathcal{O}(nP^2)$. While our results suggest the optimality of some classical algorithms, they also motivate future research in cases where the best known algorithms do not quite match the lower bounds.

Keywords: SETH · Subset sum · Scheduling · Fine-grained complexity · Pseudo-polynomial algorithms

1 Introduction

Consider the problem of working on multiple research papers. Each paper j has to go to some specific journal or conference and thus has a given due date d_j. Some papers might be more important than others, so each one has a weight w_j. In order to not get distracted, we may only work on one paper at a time and this work may not be interrupted. If a paper does not meet its due date, it is not important by how much it misses it; it is either late or on time. If it is late, we

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

L. Gąsieniec (Ed.): SOFSEM 2023, LNCS 13878, pp. 192–206, 2023.
https://doi.org/10.1007/978-3-031-23101-8_13

must pay its weight w_j. In the literature, this problem is known as $1||\sum w_jU_j$ and it is one of Karp's original 21 NP-hard problems [16]. The naming of $1||\sum w_jU_j$ and the problems referred to in the abstract will become clear when we review the three-field notation by Graham et al. [10] in Sect. 2. Even when restricted to a fixed number of identical machines, many combinations of job characteristics and objective functions lead to NP-hard problems. For this reason, a lot of effort has been put towards finding either pseudo-polynomial exact or polynomial approximation algorithms. Sticking to our problem $1||\sum w_jU_j$, where we aim to minimize the weighted number of late jobs on a single machine, there are e.g. an $\mathcal{O}(nW)$ algorithm by Lawler and Moore [21] and an FPTAS by Gens and Levner [9]. Here, W is the sum of all weights w_j and n is the number of jobs.

In recent years, research regarding scheduling has made its way towards parameterized and fine-grained complexity (see e.g. [2,12,18,26,27]), where one goal is to identify parameters that make a problem difficult to solve. If those parameters are assumed to be small, parameterized algorithms can be very efficient. Similarly, one may consider parameters like the total processing time P and examine how fast algorithms can be in terms of these parameters, while maintaining a sub-exponential dependency on n. That is our main goal in this work. Most of our lower bounds follow from a lower bound for Subset Sum:

Problem 1. Subset Sum

Instance: Items $a_1, \ldots, a_n \in \mathbb{N}$, integer target $T \in \mathbb{N}$.
Task: Decide whether there is a subset $S \subseteq [n]$ such that $\sum_{i \in S} a_i = T$.

Fine-grained running time lower bounds are often based on the Exponential Time Hypothesis (ETH) or the Strong Exponential Time Hypothesis (SETH). Intuitively, the ETH conjectures that 3-Sat cannot be solved in sub-exponential time and the SETH conjectures that the trivial running time of $\mathcal{O}(2^n)$ is optimal for k-Sat, if k tends to infinity. For details, see the original publication by Impagliazzo and Paturi [13]. A few years ago, Abboud et al. gave a beautiful reduction from k-Sat to Subset Sum [1]. Previous results based on the ETH excluded $2^{o(n)}T^{o(1)}$-time algorithms [15], while this new result based on the SETH suggests that we cannot even achieve $\mathcal{O}\left(2^{\delta n}T^{1-\varepsilon}\right)$:

Theorem 1 (Abboud et al. [1]). *For every $\varepsilon > 0$, there is a $\delta > 0$ such that* Subset Sum *cannot be solved in time $\mathcal{O}\left(2^{\delta n}T^{1-\varepsilon}\right)$, unless the SETH fails.*[1]

By revisiting many classical reductions in the context of fine-grained complexity, we transfer this lower bound to scheduling problems like $1||\sum w_jU_j$. Although lower bounds do not have the immediate practical value of an algorithm, it is clear from the results of this paper how finding new lower bounds can push research into the right direction: Our lower bound for the scheduling problem $P2|any|C_{\max}$ indicated the possibility of an $\mathcal{O}(nP)$-time algorithm, but the best known algorithm (by Du and Leung [8]) had running time $\mathcal{O}\left(nP^2\right)$. A modification of this algorithm closes this gap.

It should be noted that all lower bounds in this paper are conditional, that is, they rely on some complexity assumption. However, all of these assumptions

[1] Though it might seem unintuitive at first, it is not required that $\varepsilon < 1$.

are reasonable in the sense that a lot of effort has been put towards refuting them. And in the unlikely case that they are indeed falsified, this would have big complexity theoretical implications.

This paper is organized as follows: We first give an overview on terminology, the related lower bounds by Abboud et al. [2] and our results in Sect. 2. Then we examine scheduling problems with a single machine in Sect. 3 and problems with two or more machines in Sect. 4. Finally, we give a summary as well as open problems and promising research directions in Sect. 5.

2 Preliminaries

In this section, we first introduce the Partition problem, a special case of Subset Sum from which many of our reductions start. Then we recall common terminology from scheduling theory and finally, we give a short overview of the recent and closely related work [2] by Abboud et al. and briefly state our main results.

Throughout this paper, log denotes the base 2 logarithm. Moreover, we write $[n]$ for the set of integers from 1 to n, i.e. $[n] := \{1, \ldots, n\}$. If we consider a set of items or jobs $[n]$ and a subset $S \subseteq [n]$, we use $\bar{S} = [n] \setminus S$ to denote the complement of S. The $\tilde{\mathcal{O}}$-notation hides poly-logarithmic factors.

2.1 Subset Sum and Partition

In this work, we provide lower bounds for several scheduling problems; our main technique are *fine-grained reductions*, which are like polynomial-time reductions, but with more care for the exact sizes and running times. With these reductions, we can transfer the (supposed) hardness of one problem to another. Most of the time, our reductions start with an instance of Subset Sum or Partition and construct an instance of some scheduling problem. Partition is the special case of Subset Sum, where the sum of all items is exactly twice the target value:

Problem 2 Partition

Instance: Items $a_1, \ldots, a_n \in \mathbb{N}$.

Task: Decide whether there is a subset $S \subseteq [n]$ such that $\sum_{i \in S} a_i = \sum_{i \in \bar{S}} a_i$.

In the following, we always denote the total size of all items by $A := \sum_{i=1}^{n} a_i$ for Subset Sum and Partition. Note that we can always assume that $T \leq A$, since otherwise the target cannot be reached, even by taking all items. Moreover, in the reduction by Abboud et al. [1], A and T are quite close, in particular, we can assume that $A = \text{poly}(n)T$. Hence, if we could solve Subset Sum in time $\mathcal{O}\left(2^{\delta n} A^{1-\varepsilon}\right)$ for some $\varepsilon > 0$ and every $\delta > 0$, this would contradict Theorem 1 for large enough n. For details on this, we refer to the full version [14].

Corollary 1. *For every $\varepsilon > 0$, there is a $\delta > 0$ such that* Subset Sum *cannot be solved in time $\mathcal{O}\left(2^{\delta n} A^{1-\varepsilon}\right)$, unless the SETH fails.*

Using a classical reduction from Subset Sum to Partition that only adds two large items, we also get the following lower bound for Partition (for a detailed proof, see [14]):

Theorem 2. *For every $\varepsilon > 0$, there is a $\delta > 0$ such that* Partition *cannot be solved in time $\mathcal{O}\left(2^{\delta n} A^{1-\varepsilon}\right)$, unless the SETH fails.*

2.2 Scheduling

In all scheduling problems we consider, we are given a number of machines and a set of n jobs with processing times p_j, $j \in [n]$; our goal is to assign each job to (usually) one machine such that the resulting *schedule* minimizes some objective.[2] So these problems all have a similar structure: A machine model, some (optional) job characteristics and an objective function. This structure motivates the use of the three-field notation introduced by Graham et al. [10]. Hence, we denote a scheduling problem as a triple $\alpha|\beta|\gamma$, where α is the machine model, β is a list of (optional) job characteristics and γ is the objective function. As is usual in the literature, we leave out job characteristics like due dates that are implied by the objective function, e.g. for $1||\sum w_j U_j$. In this work, we mainly consider the decision variants of scheduling problems (as opposed to the optimization variants). In the decision problems, we are always given a threshold denoted by y and the task is to decide whether there is a solution with value at most y. Note that the optimization and the decision problems are – at least in our context – equivalent: An algorithm for the decision problem can be used to find a solution of the optimization problem with a binary search over the possible objective values (which are always integral and bounded, here). Vice versa, an algorithm for the optimization problem can also solve the decision problem.

In order to have a unified notation, given some job-dependent parameters $g_1, \ldots, g_n$ (e.g. processing times), we let $g_{\max} := \max_{i\in[n]} g_i$, $g_{\min} := \min_{i\in[n]} g_i$ and $G := \sum_{i\in[n]} g_i$. We now briefly go over the considered machine models, job characteristics and objective functions.

As the title of this work suggests, we consider problems with a fixed number of m parallel identical machines, denoted by 'Pm' if $m > 1$ or simply '1' if $m = 1$. In this setting, a job has the same processing time on every machine.

In the case of *rigid* and *moldable jobs*, each job has a given '*size*' and must be scheduled on that many machines or it may be scheduled on '*any*' number of machines, respectively, needing a possibly different (usually lower) processing time when scheduled on multiple machines. Sometimes, not all jobs are available at time 0, but instead each job j arrives at its release date 'r_j'.[3] Similarly, jobs might have deadlines d_j (i.e. due dates that may not be missed) and we must assure that '$C_j \leq d_j$' holds for every job j, where C_j is the *completion time* of j. Additionally, every job j might have a weight w_j and we are allowed to reject

[2] Depending on the scheduling problem, it may also be important in which order the jobs of a machine are scheduled or whether there are gaps between the execution of consecutive jobs.

[3] This is not to be confused with *online* scheduling; we know the r_j's in advance.

(i.e., choose not to schedule) jobs of total weight at most Q; this constraint is denoted by '$Rej \leq Q$'.[4]

The arguably most popular objective in scheduling is to minimize the so-called *makespan* '$C_{\max}$', which is the largest completion time C_j among all jobs j, i.e. the time at which all jobs are finished. In order to give the jobs different priorities, we can minimize the *total (weighted) completion time* '$\sum C_j$' ('$\sum w_j C_j$'). If there is a due date d_j for each job, we might be concerned with minimizing the *(weighted) number of late jobs* '$\sum U_j$' ('$\sum w_j U_j$'), where $U_j = 1$ if j is late, i.e. $C_j > d_j$ and $U_j = 0$ otherwise. Similar objectives are the *maximum lateness* '$L_{\max}$' and the *maximum tardiness* '$T_{\max}$' of all jobs, where the lateness L_j of job j is the (uncapped) difference $C_j - d_j$ and the tardiness T_j is the (capped) difference $\max\{C_j - d_j, 0\}$. Another objective, the *total tardiness* '$\sum T_j$', measures the tardiness of all jobs together and the *total late work* '$\sum V_j$' is the late work $V_j := \min\{p_j, C_j - d_j\}$ summed over all jobs. Both objectives may also appear in combination with weights. Lastly, if release dates r_j are present, we might be interested in minimizing the maximum flow time '$F_{\max}$', the total flow time '$\sum F_j$' or the weighted total flow time '$\sum w_j F_j$'. These objectives are similar to the previous ones; F_j, the flow time of job j, is defined as $F_j := C_j - r_j$, i.e. the time that passes between j's release and completion.

2.3 The Scheduling Lower Bounds by Abboud et al.

In their more recent work [2], Abboud et al. show lower bounds for the problems $1||\sum w_j U_j$, $1|Rej \leq Q|\sum U_j$, $1|Rej \leq Q|T_{\max}$, $1|r_j, Rej \leq Q|C_{\max}$, $P2||T_{\max}$, $P2||\sum U_j$, $P2|r_j|C_{\max}$ and $P2|level\text{-}order|C_{\max}$.[5] From those problems, only $1||\sum w_j U_j$ appears in this version; the full version [14] also contains results for $1|Rej \leq Q|\sum U_j$, $1|Rej \leq Q|T_{\max}$, $P2||T_{\max}$ and $P2||\sum U_j$. As we will see however, the results by Abboud et al. are not directly comparable to our results.

Standard dynamic programming approaches often give running times like $\mathcal{O}(nP)$; on the other hand, it is usually possible to try out all subsets of jobs, yielding an exponential running time like $\mathcal{O}(2^n \text{polylog}(P))$ (see e.g. the work by Jansen et al. [15]). The intuitive way of thinking about our lower bounds is that we cannot have the best of both worlds, i.e.: *'An algorithm cannot be sub-exponential in n and sub-linear in P at the same time.'* To be more specific, most of our lower bounds have this form: For every $\varepsilon > 0$, there is a $\delta > 0$ such that the problem cannot be solved in time $\mathcal{O}\left(2^{\delta n} P^{1-\varepsilon}\right)$.

However, note that algorithms with running time $\tilde{\mathcal{O}}(n + P)$ or $\tilde{\mathcal{O}}(n + p_{\max})$ are not excluded by our bounds, as they are not sub-linear in P. But in a setting where n and P (resp. $p_{\max}$) are roughly of the same order, such algorithms would be much more efficient than the dynamic programming approaches. In particular, they would be near-linear in n instead of quadratic. This is where

[4] This is usually denoted by $Rej \leq R$, but since we will use R for the sum of all release dates, we denote the total rejection weight by Q.

[5] In 'level-order' problems, the jobs are ordered hierarchically and all jobs of one level have to be finished before jobs of higher levels can be scheduled.

the lower bounds from the more recent paper [2] by Abboud et al. come into play, as they have the following form: There is no $\varepsilon > 0$ such that the problem can be solved in time $\tilde{\mathcal{O}}\left(n + p_{\max} n^{1-\varepsilon}\right)$, unless the $\forall\exists$-SETH fails. The $\forall\exists$-SETH is similar to the SETH, but assuming yet another assumption (the NSETH), $\forall\exists$-SETH is a strictly stronger assumption than SETH. However, these lower bounds by Abboud et al. [2] can exclude algorithms with an additive-type running time $\tilde{\mathcal{O}}\left(n + p_{\max}\right)$. Algorithms with running time $\tilde{\mathcal{O}}\left(n + p_{\max} n\right)$ may still be possible, but they would only be near-quadratic instead of near-linear in the $n \approx p_{\max}$ setting. It should be noted that our lower bounds also include parameters other than $p_{\max}$, e.g. the largest due date $d_{\max}$ or the threshold for the objective value y.

2.4 Our Results

The main contribution of this work is two-fold: On the one hand, we give plenty of lower bounds for classical scheduling problems with a fixed number of machines. These lower bounds all either rely on the ETH, SETH or the $(\min, +)$-conjecture[6] and are shown by revisiting classical reductions in the context of fine-grained complexity, i.e., we pay much attention to the parameters of the constructed instances. On the other hand, we show how the dynamic programming algorithms for $P2|any|C_{\max}$ and $P3|any|C_{\max}$ by Du and Leung [8] can be improved. Most notably, we show the following (for the precise statements, we refer to the upcoming sections):

- The algorithm by Lawler and Moore [21] is probably optimal for $1||\sum w_j U_j$ and $Pm||C_{\max}$.
- The algorithm by Lee and Uzsoy [23] is probably optimal for $P2||\sum w_j C_j$.
- The algorithm by Zhang et al. [30] for $1|Rej \leq Q|\sum w_j U_j$, the algorithm by Lawler [19] for $1||\sum T_j$ and the FPTAS by Gens and Levner [9] for $1||\sum w_j U_j$ are nearly optimal, but there is still some room for improvement.
- $P2|any|C_{\max}$ can be solved in time $\mathcal{O}(nP)$ *and* this is probably optimal.
- $P3|any|C_{\max}$ can be solved in time $\mathcal{O}\left(nP^2\right)$, which greatly improves upon the $\mathcal{O}\left(nP^5\right)$-time algorithm by Du and Leung [8].

Due to space restrictions, this version does not include the following content, which can be found in the full version [14]:

- Lower bounds for strongly NP-hard problems,
- implications of our lower bounds for other scheduling problems using classical reductions between objective functions (see Fig. 1),
- detailed correctness proofs of the classical reductions from the literature and
- proofs of some of our (less prominent or more technical) results.

[6] Under the $(\min, +)$-conjecture, the $(\min, +)$-convolution problem cannot be solved in sub-quadratic time, see [6] for details.

Note that our SETH-based lower bounds mainly show that improvements for some pseudo-polynomial algorithms are unlikely. For problems that are strongly NP-hard, pseudo-polynomial algorithms cannot exist, unless P = NP [4]. However, the lower bounds for strongly NP-hard problems may be of independent interest, e.g. in the context of parameterized algorithms.

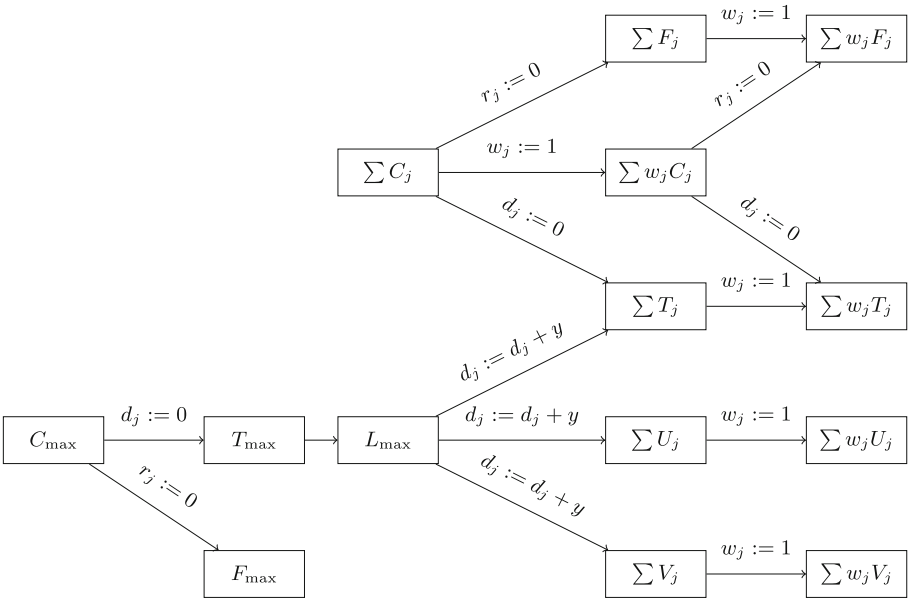

Fig. 1. Classical reductions between objective functions (see e.g. [20] and the very useful website http://schedulingzoo.lip6.fr/about.php).

3 Problems with One Machine

In this section, we consider problems on a single machine. For these problems, the main task is to order the jobs. First, consider again the problem $1||\sum w_j U_j$ of minimizing the weighted number of late jobs on a single machine. With a reduction very similar to the one by Karp [16], we get the following lower bound:[7]

Theorem 3. *For every $\varepsilon > 0$, there is a $\delta > 0$ such that $1||\sum w_j U_j$ cannot be solved in time $\mathcal{O}\left(2^{\delta n}(d_{\max} + y + P + W)^{1-\varepsilon}\right)$, unless the SETH fails.*

[7] It should be noted that some of the parameters in our lower bounds could be omitted, as they are overshadowed by others. For example, we can assume w.l.o.g. that $d_{\max} \leq P$ for $1||\sum w_j U_j$, since we can assume a schedule to be gap-less and hence due dates larger than P could be set to P. But having all the parameters in the lower bound makes the comparison with known upper bounds easier.

Proof. Let $a_1, \ldots, a_n$ be a PARTITION instance and let $T = \frac{1}{2}\sum_{i=1}^{n} a_i$. Construct an instance of $1||\sum w_j U_j$ by setting $p_j = w_j = a_j, d_j = T$ for each $j \in [n]$ and $y = T$. The idea is that the jobs corresponding to items in one of the partitions can be scheduled early (i.e. before the uniform due date T).

With this reduction, we get $N := n$ jobs. We have $P = \sum_{i=1}^{n} a_i = A$ and hence $K := d_{\max} + y + P + W = T + T + A + A = \text{poly}(n)A = n^c A$. The reduction itself takes time $\mathcal{O}(N)$. Assuming that we can solve $1||\sum w_j U_j$ in time $\mathcal{O}\left(2^{\delta N} K^{1-\varepsilon}\right)$ for some $\varepsilon > 0$ and every $\delta > 0$, we could also solve PARTITION in time:

$$\begin{aligned}\mathcal{O}(N) + \mathcal{O}\left(2^{\delta N} K^{1-\varepsilon}\right) = \mathcal{O}(n) + \mathcal{O}\left(2^{\delta n}(n^c A)^{1-\varepsilon}\right) &\leq \mathcal{O}\left(2^{\delta n} n^c A^{1-\varepsilon}\right) \\ &= \mathcal{O}\left(2^{\delta n + c\log(n)} A^{1-\varepsilon}\right) \\ &\leq \mathcal{O}\left(2^{2\delta n} A^{1-\varepsilon}\right)\end{aligned}$$

The last step holds for large enough n; for smaller n, we can solve the problem efficiently, anyway, as n is then bounded by a constant. Now, to contradict Theorem 2, we can set $\varepsilon' := \varepsilon$ and for every $\delta' > 0$, we have $\delta = \frac{\delta'}{2} > 0$. So by assumption, we can solve PARTITION in time $\mathcal{O}\left(2^{2\delta n} A^{1-\varepsilon}\right) = \mathcal{O}\left(2^{\delta' n} A^{1-\varepsilon'}\right)$. □

Using the algorithm by Lawler and Moore [21], $1||\sum w_j U_j$ is solvable in time $\mathcal{O}(nW)$ or $\mathcal{O}(n \min\{d_{\max}, P\})$. Our $\mathcal{O}\left(2^{\delta n}(d_{\max} + y + P + W)^{1-\varepsilon}\right)$-time lower bound suggests the optimality of both variants, as we cannot hope to reduce the linear dependency on W, $d_{\max}$ or P without getting a super-polynomial dependency on n. As noted above, Abboud et al. [2] exclude $\tilde{\mathcal{O}}\left(n + p_{\max} n^{1-\varepsilon}\right)$-time algorithms; Hermelin et al. [12] exclude algorithms with running time $\tilde{\mathcal{O}}\left(n + w_{\max} n^{1-\varepsilon}\right)$, $\tilde{\mathcal{O}}\left(n + w_{\max}^{1-\varepsilon} n\right)$ and $\tilde{\mathcal{O}}\left(n^{\mathcal{O}(1)} + d_{\max}^{1-\varepsilon}\right)$ (all under the stronger $\forall\exists$-SETH).

One interesting property of $1||\sum w_j U_j$ is that its straightforward formulation as an Integer Linear Program has a triangular structure that collapses to a single constraint when all due dates are equal (see e.g. Lenstra and Shmoys [25]). This shows that the problem is closely related to KNAPSACK:

Problem 3. KNAPSACK

Instance: Item values $v_1, \ldots, v_n \in \mathbb{N}$, item sizes $a_1, \ldots, a_n \in \mathbb{N}$, knapsack capacity $T \in \mathbb{N}$ and threshold y.

Task: Decide whether there is a subset S of items with $\sum_{j \in S} a_j \leq T$ and $\sum_{j \in S} v_j \geq y$.

Cygan et al. [6] conjectured that the $(\min, +)$-CONVOLUTION problem cannot be solved in sub-quadratic time (this is known as the $(\min, +)$-conjecture) and showed that this conditional lower bound transfers to KNAPSACK, excluding $\mathcal{O}\left((n+T)^{2-\delta}\right)$ algorithms. As noted by Mucha et al. [28], these results also hold when we swap the role of sizes and values. As we can discard items with too large value v_i, a lower bound depending on the largest item value $v_{\max}$ directly follows from Corollary 9.6 in [28]:

Corollary 2. *For any constant $\delta > 0$, there is no $\mathcal{O}\left((n + v_{\max})^{2-\delta}\right)$-time exact algorithm for* Knapsack*, unless the* $(\min, +)$*-conjecture fails.*

We show that the conditional hardness of Knapsack transfers to $1||\sum w_j U_j$:

Theorem 4. *For any constant $\delta > 0$, the existence of an exact algorithm for $1||\sum w_j U_j$ with running time $\mathcal{O}\left((n + w_{\max})^{2-\delta}\right)$ refutes the* $(\min, +)$*-conjecture.*

Proof. We give a reduction from Knapsack to $1||\sum w_j U_j$. Consider an instance $v_1, \ldots, v_n$, $a_1, \ldots, a_n$, T, y of Knapsack. We construct jobs with $p_j = a_j$, $w_j = v_j$ and $d_j = T$ for every $j \in [n]$. The threshold is set to $y' = \sum_{j=1}^{n} v_j - y$.

Suppose that there is an $\mathcal{O}\left((n + w_{\max})^{2-\delta}\right)$-time algorithm for $1||\sum w_j U_j$. Since $w_{\max} = v_{\max}$ in the reduction and the reduction takes time $\mathcal{O}(n)$, we could then solve Knapsack in time $\mathcal{O}(n) + \mathcal{O}\left((n + w_{\max})^{2-\delta}\right) = \mathcal{O}\left((n + v_{\max})^{2-\delta}\right)$, which is a contradiction to Corollary 2, unless the $(\min, +)$-conjecture fails. □

Lower bounds such as this one also imply lower bounds for approximation schemes, as setting the accuracy parameter ε small enough yields an exact solution. The above result implies the following (see the full version [14] for the proof):

Corollary 3. *For any constant $\delta > 0$, the existence of an $\mathcal{O}\left((n + \frac{1}{2n\varepsilon})^{2-\delta}\right)$-time approximation scheme for the optimization version of $1||\sum w_j U_j$ refutes the* $(\min, +)$*-conjecture.*

As the currently fastest FPTAS by Gens and Levner [9] has running time $\mathcal{O}\left(n^2(\log(n) + \frac{1}{\varepsilon})\right)$, there is still a small gap. This relation between exact and approximation algorithms might also be an interesting subject of further investigation, as many other scheduling problems admit approximation schemes and exact lower bounds.

We wish to mention two other results that follow from examining classical reductions, the proofs of which can also be found in the full version [14]. The first result concerns $1||\sum T_j$:

Theorem 5. *For every $\varepsilon > 0$, there is a $\delta > 0$ such that $1||\sum T_j$ cannot be solved in time $\mathcal{O}\left(2^{\delta n} P^{1-\varepsilon}\right)$, unless the SETH fails.*

There is an $\mathcal{O}\left(n^4 P\right)$-time algorithm by Lawler [19] and while we can derive no statement about the exponent of n, our lower bound suggests that an improvement of the linear factor P is unlikely without getting a super-polynomial dependency on n. We have a similar situation for the problem $1|Rej \leq Q|C_{\max}$:

Theorem 6. *For every $\varepsilon > 0$, there is a $\delta > 0$ such that $1|Rej \leq Q|C_{\max}$ cannot be solved in time $\mathcal{O}\left(2^{\delta n}(y + P + Q + W)^{1-\varepsilon}\right)$, unless the SETH fails.*

The lower bound can also be shown to hold for $1|Rej \leq Q|\sum w_j U_j$ using reductions between objective functions (see Fig. 1) and this problem can be solved in time $\mathcal{O}(nQP)$ with the algorithm by Zhang et al. [30]. This almost matches our lower bound: An algorithm with running time $\mathcal{O}(n(Q + P))$ might still be possible, for example.

4 Problems with Multiple Machines

We now turn our attention to problems on two or more machines. For **standard jobs**, a straightforward reduction from Partition yields the following result (for a formal proof, see [14]):

Theorem 7. *For every $\varepsilon > 0$, there is a $\delta > 0$ such that $P2||C_{\max}$ cannot be solved in time $\mathcal{O}\left(2^{\delta n}(y+P)^{1-\varepsilon}\right)$, unless the SETH fails.*

This lower bound also applies to the harder objectives (e.g. $T_{\max}$) and in particular to $P2||\sum w_jU_j$ (using the reductions in Fig. 1); the dynamic program by Lawler and Moore [21] (which is also sometimes attributed to Rothkopf [29]) solves most common objectives like $C_{\max}$ and $T_{\max}$ in time $\mathcal{O}(ny)$ but needs $\mathcal{O}\left(ny^2\right)$ for $P2||\sum w_jU_j$ (see [25], in particular exercise 8.10). So the gap is likely closed in the $C_{\max}$, $T_{\max}$, ... cases, but there is still a factor-y-gap for the $\sum w_jU_j$-objective.

In general, the dynamic program by Lawler and Moore [21] solves $Pm||C_{\max}$ in a running time of $\mathcal{O}\left(nmy^{m-1}\right) \leq \mathcal{O}\left(nmP^{m-1}\right)$. Our matching lower bound for $m = 2$ gives rise to the question whether the running time is optimal for general $m > 1$. Chen et al. [5] showed a $2^{\mathcal{O}\left(m^{\frac{1}{2}-\delta}\sqrt{|I|}\right)}$-time lower bound for $Pm||C_{\max}$ and with a careful analysis, one can also show the following lower bound (see the full version [14] for a proof):

Theorem 8. *There is no $\mathcal{O}\left(nmP^{o\left(\frac{m}{\log^2(m)}\right)}\right)$-time algorithm for $Pm||C_{\max}$, unless the ETH fails.*

So the algorithm by Lawler and Moore [21] is indeed almost optimal, as we can at best hope to shave off logarithmic factors in the exponent (assuming the weaker assumption ETH). Since the algorithm not only works for $C_{\max}$, one might ask whether we can find similar lower bounds for other objectives as well. For most common objective functions, we answer this question positively using the reductions in Fig. 1 (see the full version [14]), but it remains open for $\sum w_jC_j$. Note that the unweighted $Pm||\sum C_j$ is polynomial-time solvable [3].

An alternative dynamic program by Lee and Uzsoy [23] solves $Pm||\sum w_jC_j$ in time $\mathcal{O}\left(mnW^{m-1}\right)$. In order to get a matching lower bound (i.e. one that depends on the weights) for $m = 2$, we examine another classical reduction:

Theorem 9. *For every $\varepsilon > 0$, there is a $\delta > 0$ such that $P2||\sum w_jC_j$ cannot be solved in time $\mathcal{O}\left(2^{\delta n}(\sqrt{y}+P+W)^{1-\varepsilon}\right)$, unless the SETH fails.*

Proof. We show that the lower bound for Partition can be transferred to $P2||\sum w_jC_j$ using the reduction by Lenstra et al. [24] and Bruno et al. [3].

Given a Partition instance $a_1, \ldots, a_n$, we construct a $P2||\sum w_jC_j$ instance in the following way: Define $p_j = w_j = a_j$ for all $j \in [n]$ and set the limit $y = \sum_{1\leq i\leq j\leq n} a_ja_i - \frac{1}{4}A^2$. Of course, the idea of the reduction is that the limit y forces the jobs to be equally distributed among the two machines (regarding the processing time).

Assume that there is an algorithm that solves an instance of $P2||\sum w_jC_j$ in time $\mathcal{O}\left(2^{\delta N}K^{1-\varepsilon}\right)$ for some $\varepsilon > 0$ and every $\delta > 0$, where $N := n$ and $K := \sqrt{y} + P + W$. By the choice of y, we can see that

$$y = \sum_{1\leq i\leq j\leq n} a_j a_i - \frac{1}{4}A^2 \leq \left(\sum_{j\in[n]} a_j\right)^2 - \frac{1}{4}A^2 = \frac{3}{4}A^2 = \mathcal{O}\left(A^2\right).$$

Since $w_j = p_j = a_j$, we also have $P = W = A$. Hence, we have $K = \sqrt{y}+P+W = \mathcal{O}\left(A + A + A\right) = \mathcal{O}\left(A\right)$ and an algorithm with running time

$$\mathcal{O}\left(2^{\delta N}K^{1-\varepsilon}\right) = \mathcal{O}\left(2^{\delta n}\mathcal{O}\left(A\right)^{1-\varepsilon}\right) = \mathcal{O}\left(2^{\delta n}c^{1-\varepsilon}A^{1-\varepsilon}\right) = \mathcal{O}\left(2^{\delta n}A^{1-\varepsilon}\right)$$

would contradict the lower bound for PARTITION from Theorem 2. Here, c covers the constants in the $\mathcal{O}$-term and the running time $\mathcal{O}\left(N\right)$ of the reduction vanishes. □

So the $\mathcal{O}\left(nW\right)$-time algorithm by Lee and Uzsoy [23] is probably optimal for $P2||\sum w_jC_j$, as we cannot hope to reduce the linear dependency on W without getting a super-polynomial dependency on n.

We briefly turn our attention towards **rigid jobs**. Clearly, $P2|size|C_{\max}$ is a generalization of $P2||C_{\max}$ (the latter problem simply does not have two-machine jobs), so we get the following lower bound (for a formal proof, see the full version [14]):

Theorem 10. *For every $\varepsilon > 0$, there is a $\delta > 0$ such that $P2|size|C_{\max}$ cannot be solved in time $\mathcal{O}\left(2^{\delta n}(y+P)^{1-\varepsilon}\right)$, unless the SETH fails.*

Similarly, the algorithm by Lawler and Moore [21] can be used to find a feasible schedule for the one-machine jobs and the two-machine jobs can be scheduled at the beginning. This gives an $\mathcal{O}\left(ny\right)$-time algorithm for $P2|size|C_{\max}$, and the linear dependency on y cannot be improved without getting a super-polynomial dependency on n, unless the SETH fails. For other objectives, the problem quickly becomes more difficult: Already $P2|size|L_{\max}$ is strongly NP-hard, as well as $P2|size|\sum w_jC_j$ (for both results, see Lee and Cai [22]). It is still open whether the unweighted version $P2|size|\sum C_j$ is also strongly NP-hard or whether there is a pseudo-polynomial algorithm; this question has already been asked by Lee and Cai [22], more than 20 years ago.

It is not hard to see that the hardness of $P2||C_{\max}$ also transfers to **moldable jobs** (i.e. $P2|any|C_{\max}$); we simply create an instance where it does not make sense to schedule any of the jobs on two machines (again, for a formal proof, see the full version [14]):

Theorem 11. *For every $\varepsilon > 0$, there is a $\delta > 0$ such that $P2|any|C_{\max}$ cannot be solved in time $\mathcal{O}\left(2^{\delta n}(y+P)^{1-\varepsilon}\right)$, unless the SETH fails.*

The problems $P2|any|C_{\max}$ and $P3|any|C_{\max}$ can be solved via dynamic programming, as shown by Du and Leung [8] (a nice summary is given in the book by Drozdowski [7]). We show that these programs can be improved to match our new lower bound for the two-machine case:

Theorem 12. *The problem $P2|any|C_{\max}$ can be solved in time $\mathcal{O}(nP)$ via dynamic programming.*

Proof. Assume that we are given processing times $p_j(k)$, indicating how long it takes to run job j on k machines. The main difficulty is to decide whether a job is to be processed on one or on two machines. Our dynamic program fills out a table $F(j,t)$ for every $j \in [n]$ and $t \in [y]$, where the entry $F(j,t)$ is the minimum load we can achieve on machine 2, while we schedule all the jobs in $[j]$ and machine 1 has load t. To fill the table, we use the following recurrence formula:

$$F(j,t) = \min \begin{cases} F(j-1, t-p_j(1)) \\ F(j-1,t) + p_j(1) \\ F(j-1, t-p_j(2)) + p_j(2) \end{cases}$$

Intuitively speaking, job j is executed on machine 1 in the first case, on machine 2 in the second case and on both machines in the third case. The initial entries of the table are $F(0,0) = 0$ and $F(0,t) = \infty$ for every $t \in [y]$.

There are $ny \leq n \sum_{j=1}^{n} \max\{p_j(1), p_j(2)\} = \mathcal{O}(nP)$ entries we have to compute.[8] Then, we can check for every $t \in [y]$ whether $F(n,t) \leq y$. If we find such an entry, this directly corresponds to a schedule with makespan at most y, so we can accept. Otherwise, there is no such schedule and we can reject. The actual schedule can be obtained by traversing backwards through the table; alternatively, we can store the important bits of information while filling the table (this works exactly like, e.g., in the standard knapsack algorithm). Note that we might have to reorder the jobs such that the jobs executed on two machines are run in parallel. But it can be easily seen that all two-machine jobs can be executed at the beginning of the schedule. Computing the solution and reordering does not change the running time in $\mathcal{O}$-notation, so we get an $\mathcal{O}(nP)$ algorithm. □

As Theorem 11 shows, improving the dependency on P to sub-linear is only possible if we get a super-polynomial dependency on n, unless the SETH fails. Using a similar recurrence formula and the fact that information about an optimal placement of jobs directly leads to an optimal schedule (i.e. there is a canonical schedule), one can show a similar result for $P3|any|C_{\max}$ (see the full version [14] for a proof):

Theorem 13. *The problem $P3|any|C_{\max}$ can be solved in time $\mathcal{O}\left(n^2P\right)$ via dynamic programming.*

This improves upon the $\mathcal{O}\left(nP^5\right)$-algorithm by Du and Leung [8]. Even though the same approach could be applied to an arbitrary number of machines m in time $\mathcal{O}\left(nmP^{m-1}\right)$, the strong NP-hardness of $Pm|any|C_{\max}$ for $m \geq 4$ shows that the information on which machine each job is scheduled is not enough to directly construct an optimal schedule in those cases, unless P=NP (see Henning et al. [11] as well as Du and Leung [8]).

[8] The precise definition of P in this context does not matter for the running time in $\mathcal{O}$-notation; we can either add $p_j(1)$ *and* $p_j(2)$ to the sum or just the larger of the two.

5 Conclusion

In this work, we examined the complexity of scheduling problems with a fixed number of machines. Our conditional lower bounds indicate the optimality of multiple well-known classical algorithms. For the problems $P2|any|C_{\max}$ and $P3|any|C_{\max}$, we managed to improve the currently best known algorithm, closing the gap in the case of two machines.

As we have seen at the example of $1||\sum w_j U_j$, lower bounds for exact algorithms can be quite easily used to obtain lower bounds for approximation schemes. We strongly believe that the same technique can be used for other problems, either to show tightness results or to indicate room for improvement.

For exact algorithms, there is a number of open problems motivated by our results: First of all, there is still a gap between our lower bound for $Pm||C_{\max}$ (and other objectives) and the algorithm by Lawler and Moore [21]. So an interesting question is where the 'true' complexity lies between $m-1$ and $o\left(\frac{m}{\log^2(m)}\right)$ in the exponent. Zhang et al. give an $\mathcal{O}\left(n(r_{\max}+P)\right)$-time algorithm for $1|r_j, Rej \leq Q|C_{\max}$ in their work [30]. Since $r_{\max}+P \geq y$ w.l.o.g., it would be interesting to find an $\mathcal{O}\left(2^{\delta n}(r_{\max}+P)^{1-\varepsilon}\right)$ or $\mathcal{O}\left(2^{\delta n}y^{1-\varepsilon}\right)$ lower bound for this problem. As noted by Lenstra and Shmoys [25], the algorithm by Lawler and Moore [21] cannot be improved to $\mathcal{O}\left(mny^{m-1}\right)$ for the objective $\sum w_j U_j$. So this algorithm would be quadratic in y for two machines, while our lower bound excludes anything better than linear (and still polynomial in n). Hence, it would be interesting to see whether there is a different algorithm with running time $\mathcal{O}\left(ny\right)$. Similarly, there is an algorithm for $1|Rej \leq Q|\sum w_j U_j$ with running time $\mathcal{O}\left(nQP\right)$ [30], while our lower bound suggests that an $\mathcal{O}\left(n(Q+P)\right)$-time algorithm could be possible.

On another note, it would be interesting to extend the sub-quadratic equivalences by Cygan et al. [6] and Klein [17] to scheduling problems. Finally, the question by Lee and Cai [22] whether $P2|size|\sum C_j$ is strongly NP-hard or not is still open since 1999.

Acknowledgements. The authors wish to thank Sebastian Berndt, Max Deppert, Sören Domrös, Lena Grimm, Leonie Krull, Marten Maack, Niklas Rentz and anonymous reviewers for very helpful comments and ideas.

References

1. Abboud, A., Bringmann, K., Hermelin, D., Shabtay, D.: Seth-based lower bounds for subset sum and bicriteria path. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, 6–9 January, 2019, pp. 41–57 (2019). https://doi.org/10.1137/1.9781611975482.3
2. Abboud, A., Bringmann, K., Hermelin, D., Shabtay, D.: Scheduling lower bounds via and subset sum. J. Comput. Syst. Sci. **127**, 29–40 (2022). https://doi.org/10.1016/j.jcss.2022.01.005

3. Bruno, J.L., Coffman, Jr., E.G., Sethi, R.: Scheduling independent tasks to reduce mean finishing time. Commun. ACM **17**(7), 382–387 (1974). https://doi.org/10.1145/361011.361064
4. Chen, B., Potts, C.N., Woeginger, G.J.: A review of machine scheduling: Complexity, algorithms and approximability. In: Du, D.Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization: Volume 1–3, pp. 1493–1641. Springer, US, Boston, MA (1998). https://doi.org/10.1007/978-1-4613-0303-9_25
5. Chen, L., Jansen, K., Zhang, G.: On the optimality of approximation schemes for the classical scheduling problem. In: Chekuri, C. (ed.) Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, 5–7 January, 2014. pp. 657–668. SIAM (2014). https://doi.org/10.1137/1.9781611973402.50
6. Cygan, M., Mucha, M., Wundefinedgrzycki, K., Włodarczyk, M.: On problems equivalent to (min,+)-convolution. ACM Trans. Algorithms **15**(1) (2019). https://doi.org/10.1145/3293465
7. Drozdowski, M.: Scheduling for Parallel Processing. Springer Publishing Company, Incorporated, 1st edn. (2009). https://doi.org/10.1007/978-1-84882-310-5
8. Du, J., Leung, J.Y.T.: Complexity of scheduling parallel task systems. SIAM J. Discret. Math. **2**(4), 473–487 (1989). https://doi.org/10.1137/0402042
9. Gens, G., Levner, E.: Fast approximation algorithm for job sequencing with deadlines. Discret. Appl. Math. **3**(4), 313–318 (1981). https://doi.org/10.1016/0166-218X(81)90008-1
10. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.H.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer, P., Johnson, E., Korte, B. (eds.) Discrete Optimization II, Annals of Discrete Mathematics, vol. 5, pp. 287–326. Elsevier (1979). https://doi.org/10.1016/S0167-5060(08)70356-X
11. Henning, S., Jansen, K., Rau, M., Schmarje, L.: Complexity and Inapproximability Results for Parallel Task Scheduling and Strip Packing. Theory of Computing Systems **64**(1), 120–140 (2019). https://doi.org/10.1007/s00224-019-09910-6
12. Hermelin, D., Molter, H., Shabtay, D.: Minimizing the weighted number of tardy jobs via (max,+)-convolutions (2022). https://doi.org/10.48550/ARXIV.2202.06841
13. Impagliazzo, R., Paturi, R.: On the complexity of k-sat. J. Comput. Syst. Sci. **62**(2), 367–375 (2001). https://doi.org/10.1006/jcss.2000.1727
14. Jansen, K., Kahler, K.: On the complexity of scheduling problems with a fixed number of parallel identical machines (2022). https://doi.org/10.48550/ARXIV.2202.07932, https://arxiv.org/abs/2202.07932
15. Jansen, K., Land, F., Land, K.: Bounding the running time of algorithms for scheduling and packing problems. SIAM J. Discret. Math. **30**(1), 343–366 (2016). https://doi.org/10.1137/140952636
16. Karp, R.M.: Reducibility among combinatorial problems. Complexity of Computer Computations (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
17. Klein, K.M.: On the Fine-Grained Complexity of the Unbounded SubsetSum and the Frobenius Problem, pp. 3567–3582. SIAM (2022). https://doi.org/10.1137/1.9781611977073.141
18. Knop, D., Koutecký, M.: Scheduling meets n-fold integer programming. J. Sched. **21**(5), 493–503 (2017). https://doi.org/10.1007/s10951-017-0550-0

19. Lawler, E.L.: A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. In: Hammer, P., Johnson, E., Korte, B., Nemhauser, G. (eds.) Studies in Integer Programming, Annals of Discrete Mathematics, vol. 1, pp. 331–342. Elsevier (1977). https://doi.org/10.1016/S0167-5060(08)70742-8
20. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H., Shmoys, D.B.: Chapter 9 sequencing and scheduling: algorithms and complexity. In: Logistics of Production and Inventory, Handbooks in Operations Research and Management Science, vol. 4, pp. 445–522. Elsevier (1993). https://doi.org/10.1016/S0927-0507(05)80189-6
21. Lawler, E.L., Moore, J.M.: A functional equation and its application to resource allocation and sequencing problems. Manage. Sci. **16**(1), 77–84 (1969). https://doi.org/10.1287/mnsc.16.1.77
22. Lee, C.Y., Cai, X.: Scheduling one and two-processor tasks on two parallel processors. IIE Trans. **31**(5), 445–455 (1999). https://doi.org/10.1080/07408179908969847
23. Lee, C.Y., Uzsoy, R.: A new dynamic programming algorithm for the parallel machines total weighted completion time problem. Operations Research Letters 11(2), 73–75 (mar 1992). https://doi.org/10.1016/0167-6377(92)90035-2
24. Lenstra, J.K., Rinnooy Kan, A.H., Brucker, P.: Complexity of machine scheduling problems. Ann. Discrete Math. **1**, 343–362 (1977). https://doi.org/10.1016/S0167-5060(08)70743-X
25. Lenstra, J.K., Shmoys, D.B.: Elements of scheduling (2020). https://doi.org/10.48550/ARXIV.2001.06005
26. Mnich, M., van Bevern, R.: Parameterized complexity of machine scheduling: 15 open problems. Comput. Oper. Res. **100**, 254–261 (2018). https://doi.org/10.1016/j.cor.2018.07.020
27. Mnich, M., Wiese, A.: Scheduling and fixed-parameter tractability. Mathematical Programming **154**(1–2), 533–562 (dec 2015). https://doi.org/10.1007/s10107-014-0830-9
28. Mucha, M., Wundefinedgrzycki, K., Włodarczyk, M.: A subquadratic approximation scheme for partition. In: Chan, T.M. (ed.) Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, pp. 70–88. Society for Industrial and Applied Mathematics, USA (2019). https://doi.org/10.1137/1.9781611975482.5
29. Rothkopf, M.H.: Scheduling independent tasks on parallel processors. Manage. Sci. **12**(5), 437–447 (1966). https://doi.org/10.1287/mnsc.12.5.437
30. Zhang, L., Lu, L., Yuan, J.: Single-machine scheduling under the job rejection constraint. Theoret. Comput. Sci. **411**(16–18), 1877–1882 (2010). https://doi.org/10.1016/j.tcs.2010.02.006