INSTITUT FÜR INFORMATIK

Bounding the Running Time of Algorithms for Scheduling and Packing Problems

Klaus Jansen, Felix Land, Kati Land

Bericht Nr. 1302
April 2013
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Bounding the Running Time of Algorithms for Scheduling and Packing Problems

Klaus Jansen, Felix Land, Kati Land

Bericht Nr. 1302
April 2013
ISSN 2192-6247

e-mail: {kj,fku,kla}@informatik.uni-kiel.de

Abstract

We investigate the implications of the exponential time hypothesis on algorithms for scheduling and packing problems. Our main focus is to show tight lower bounds on the running time of these algorithms. For exact algorithms we investigate the dependence of the running time on the number n of items (for packing) or jobs (for scheduling). We show that many of these problems, including SUBSETSUM, KNAPSACK, BINPACKING, $\langle \mathrm{P2}\,|\,|\,C_{\max}\rangle$, and $\langle \mathrm{P2}\,|\,|\,\sum w_j C_j\rangle$, have a lower bound of $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$. We also develop an algorithmic framework that is able to solve a large number of scheduling and packing problems in time $2^{\mathrm{O}(n)} \times \|I\|^{\mathrm{O}(1)}$. Finally, we show that there is no PTAS for MULTIPLEKNAPSACK and 2D-KNAPSACK with running time $2^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)}$ and $n^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)}$.

1 Introduction

Classical complexity theory allows us to rule out polynomial time algorithms for decision and optimization problems. Often the preferred way for dealing with NP-hard problems are approximate algorithms and heuristics. In recent years however, the interest in super-polynomial exact algorithms has increased. A big problem is that, under the usual assumption $\mathsf{P} \neq \mathsf{NP}$, we cannot know what super-polynomial running times are possible for these problems.

A stronger assumption was introduced by Impagliazzo, Paturi, and Zane, the *Exponential Time Hypothesis* (ETH). The subject of the ETH is the satisfiability problem 3-SAT In contrast to classical complexity theory the running time assumed in the ETH not only depends on the length $\|\varphi\|$ of the instance, but on a special parameter of the instance, the number n of variables.

Conjecture 1.1 (Exponential Time Hypothesis [15])**.** There is positive real δ such that 3-SAT cannot be decided in time $2^{\delta n} \times \|\varphi\|^{\mathrm{O}(1)}$.

Another way to formulate the conjecture is that 3-SAT with parameter n has no *sub-exponential* algorithm. Here, we follow the notation of Flum and Grohe [7]: a function f is called sub-exponential if $f(n) = \mathrm{o}(n)$, where $f = \mathrm{o}(g)$ if there is a non-decreasing, unbounded function μ such that $g(n) \leq \frac{f(n)}{\mu(n)}$.

The ETH can be used to show lower bounds on the running time of algorithms for other problems by the use of *strong reductions*, i.e. reductions which increase the parameter at most linearly [15]. Another important result is implied by the Sparsification Lemma due to Impagliazzo, Paturi, and Zane [16]: Under assumption of the ETH there is no algorithm that decides 3-SAT with m clauses in time $2^{\mathrm{o}(m)} \times \|\varphi\|^{\mathrm{O}(1)}$. This allows us to parametrize by the number of clauses.

Our main focus in this paper are consequences of the ETH for scheduling and packing problems. We investigate the dependence of the running time on the number

of jobs respectively the number of items, which we will denote by n. We also develop algorithms that are able to solve a broad class of scheduling and packing problems and whose running time matches the lower bound for many problems. We will first concentrate on SUBSETSUM and related problems. These will then be used to show bounds on other problems.

Notation We use the notation $f(S) = \sum_{s \in s} f(s)$ for any function f and any subset S of the domain of f throughout the paper. For a minimization or maximization problem and $\alpha > 1$, an algorithm A is called α-approximate if $A(I) \leq \alpha \operatorname{OPT}(I)$ or $A(I) \geq \frac{1}{\alpha} \operatorname{OPT}(I)$ holds for each instance I, respectively.

Known Results. There is a large number of lower bounds based on the ETH, mostly in the area of graph problems. For example it is known that CLIQUE (and the equivalent INDEPENDENTSET) cannot be decided in time $2^{o(n)} \times \|I\|^{O(1)}$ [16]. For a good survey of these results and useful techniques we refer to the work of Lokshtanov, Marx, and Saurabh [23]. Only few lower bounds have been obtained for scheduling and packing problems: Chen et al. [3] showed that precedence constrained scheduling on m machines cannot be decided in time $f(m)\|I\|^{o(m)}$ and set packing cannot be decided in time $f(k)\|I\|^{o(k)}$, where k is the size of the packing. Kulik and Shachnai [19] observed that sized subset sum, where k is the size of set to be found, cannot be decided in time $f(k)\|I\|^{o\left(\sqrt{k}\right)}$ and used this result to show that there is no PTAS for the 2-dimensional vectorial knapsack problem with running time $f(\varepsilon)\|I\|^{o(\sqrt{\frac{1}{\varepsilon}})}$. These results are actually based on the assumption that not all problems in SNP are solvable in sub-exponential time. Since 3-SAT $\in$ SNP, this assumption in weaker than the ETH [27]. Pătraşcu and Williams [28] showed a lower bound of $n^{o(k)}$ for sized subset sum, even when the encoding length of the item sizes is bounded by $O(d \log n)$. Finally, Jansen et al. [18] proved that bin packing into m bins cannot be solved in time $f(m)\|I\|^{o(m/\log m)}$ when the item sizes are encoded in unary.

Exact algorithms for $\langle Pm \mid \mid C_{\max} \rangle$ with $2 \leq m \leq 4$ that have running times $\sqrt{2}^n$, $\sqrt{3}^n$ and $(1+\sqrt{2})^n$ were developed by Lenté et al. [22]. BINPACKING can be solved in time $nB2^n$ [8] or $n^{O(m)}2^{O(m\sqrt{\|I\|})}$ [25], where m is the number of bins. SUBSETSUM, PARTITION and KNAPSACK can all be solved in time $2^{\frac{n}{2}} \times \|I\|^{O(1)}$ [8, 14] and $2^{O\left(\sqrt{\|I\|}\right)}$ [25, 26].

New Results and Organization In Sect. 2 we investigate exact algorithms for SUBSETSUM and related problems, including PARTITION, BINPACKING, and MULTIPROCESSORSCHEDULING. We prove the lower bounds $2^{o(n)} \times \|I\|^{O(1)}$ and $2^{o(\sqrt{\|I\|})}$ for these problems. In Sect. 3 we give a lower bound of $2^{o(n)} \times \|I\|^{O(1)}$ for different

types of scheduling problems. We present an algorithmic framework in Sect. 4 that is able to solve nearly all problems mentioned in Sects. 2 and 3 in time $2^{\mathrm{O}(n)} \times \|I\|^{\mathrm{O}(1)}$, showing that the corresponding bounds are tight. Finally, in Sect. 5 we consider approximation schemes for knapsack problems. We prove that there are no PTAS for MULTIPLEKNAPSACK and 2D-KNAPSACK with running times $2^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)}$ and $n^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)}$, respectively.

2 The Subset Sum Family

In this section we will prove tight lower bounds on the running time of algorithms for several problems related to SUBSETSUM and PARTITION, when parametrized by the number n of items or the input size $\|I\|$.

2.1 Lower Bounds for Subset Sum and Partition

Wegener [30] presented a chain of reductions from 3-SAT to PARTITION via the subset sum problem. We will omit the proofs of correctness (they can be found in Appendix B.1) and only give a brief description of the construction.

From 3-SAT **to** SUBSETSUM**.** Denote the variables by $x_1, \ldots, x_n$ and the clauses by $C_1, \ldots, C_m$. For each variable x_i we create two items a_i and b_i with

$$\mathrm{s}(a_i) = \sum_{\substack{j \in [m] \\ x_i \in C_j}} 10^{n+j-1} + 10^{i-1} \qquad \text{and} \qquad \mathrm{s}(b_i) = \sum_{\substack{j \in [m] \\ \bar{x}_i \in C_j}} 10^{n+j-1} + 10^{i-1}.$$

These numbers have at most $n + m$ digits when encoded in base 10. Additionally we create two dummy items c_j and d_j for each clause C_j with $\mathrm{s}(c_j) = \mathrm{s}(d_j) = 10^{n+j-1}$. The item set is $A = \{a_i, b_i \mid i \in [n]\} \cup \{c_j, d_j \mid j \in [m]\}$ The target value is

$$B = \sum_{j=1}^{m} 3 \times 10^{n+j-1} + \sum_{i=1}^{n} 10^{i-1}.$$

In total the instance $I = (A, B)$ has $2n + 2m$ items, hence the reduction is strong.

From SUBSETSUM **to** PARTITION**.** Let (A, B) be an instance of SUBSETSUM. First assume that $\mathrm{s}(A) \geq B$, otherwise we can output a trivial no-instance. We introduce two new items p and q with $\mathrm{s}(p) = 2\,\mathrm{s}(A) - B$ and $\mathrm{s}(q) = \mathrm{s}(A) + B$. The instance of PARTITION is $A' = A \cup \{p, q\}$. Note that $\mathrm{s}(p) \in \mathbb{N}$ because $\mathrm{s}(A) \geq B$.

Theorem 2.1 (A proof can be found in Appendix B.1)**.** *The problems* PARTITION *and* SUBSETSUM *cannot be decided in time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$*, unless the ETH fails.*

The above bounds are asymptotically tight: A naïve enumeration algorithm solves both problems by testing all 2^n subsets of A in time $2^n \times \|I\|^{\mathrm{O}(1)}$. The fastest known algorithms have asymptotic running time $2^{\frac{n}{2}} \times \|I\|^{\mathrm{O}(1)}$ [14].

2.2 Implications for Scheduling and Packing

2.2.1 Packing in One Bin.

A generalization of SUBSETSUM is the well-known knapsack problem.

Theorem 2.2 (A proof can be found in Appendix B.2)**.** *There is no algorithm deciding* 0-1-INTEGERPROGRAMMING *(even for one constraint and only positive coefficients) or* KNAPSACK *in time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$*, unless the ETH fails.*

These results are again asymptotically tight.

2.2.2 Bin Packing and Multiprocessor Scheduling.

Another fundamental packing problem is BINPACKING. The decision problem asks if the given items fit into a given number of bins and is known to be strongly NP-hard [10]. Even the case where the number m of bins is a fixed constant, called m-BINPACKING, remains weakly NP-hard [21]. This result originates from the hardness of PARTITION, which is equivalent to 2-BINPACKING with $B = \frac{1}{2}\,\mathrm{s}(A)$. Marx [24] observed that the gap creation technique that is commonly used to show inapproximability can be used in context of the ETH. In combination with Theorem 2.1 we obtain:

Theorem 2.3. *For* $\alpha < 2$ *there is no* α*-approximate algorithm for* BINPACKING *and no exact algorithm for* 2-BINPACKING *with running time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$*, unless the ETH fails.*

The simplest variant of scheduling is the multiprocessor scheduling problem MPS. It asks if there is a schedule of the given jobs on m machines that finishes within a given deadline D and is equivalent to BINPACKING.

Theorem 2.4. *There is no algorithm deciding* MPS *in time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$*, unless the ETH fails. This also holds for a fixed number* $m \geq 2$ *of machines.*

We present algorithms for MPS and BINPACKING with running time $2^{\mathrm{O}(n)} \times \|I\|^{\mathrm{O}(1)}$ in Sect. 4, which closes the gap between upper and lower bounds.

2.3 Input Length as Complexity Measure

When the running time is measured in the encoding length of the input the fastest known algorithms for SUBSETSUM, PARTITION, KNAPSACK and m-BINPACKING have running time $2^{\mathrm{O}(\sqrt{\|I\|})}$ [25, 26].

Theorem 2.5. SUBSETSUM, PARTITION, KNAPSACK *and* m-BINPACKING *with* $m \geq 2$ *cannot be decided in time* $2^{\mathrm{o}(\sqrt{\|I\|})}$*, unless the ETH fails.*

Proof. Consider an instance of SUBSETSUM as constructed by the reduction for Theorem 2.1. It contains $2n + 2m$ numbers, and each can be encoded (in base 10) with at most $n+m$ digits. Because we can assume that $n = \mathrm{O}(m)$ we know that $\|I\| = \mathrm{O}(m^2)$. If an algorithm for SUBSETSUM with running time $2^{\mathrm{o}(\sqrt{\|I\|})}$ existed, one could use it to solve 3-SAT in time $2^{\mathrm{o}(m)} \times \|\varphi\|^{\mathrm{O}(1)}$. The reductions to the other problems do not increase the encoding length of the instance significantly. □

2.4 Special Cases with Size Restrictions

If φ is some predicate on the instances of SUBSETSUM or PARTITION, we denote the problem restricted to instances for which the predicate is TRUE by SUBSETSUM-φ or PARTITION-φ, respectively.

We first restrict SUBSETSUM to instances (A, B) with the following property: If a subset $S \subseteq A$ with $\mathrm{s}(S) = B$ exists, then it contains exactly half of the elements, or more formally the following predicate φ holds:

$$\varphi((A,B)) \iff \forall S \subseteq A\colon \left(\mathrm{s}(S) = B \implies |S| = \tfrac{|A|}{2}\right).$$

Lemma 2.6 (A proof can be found in Appendix B.3)**.** *There is no algorithm that decides* SUBSETSUM-φ *in time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$*, unless the ETH fails.*

Proofsketch. We give a strong reduction from SUBSETSUM. Let (A, B) be an instance of SUBSETSUM. For each item $a \in A$ we construct two items a_1 and a_2 with $\mathrm{s}(a_1) = 2n\,\mathrm{s}(a) + 1$ and $\mathrm{s}(a_2) = 1$, and let $A' = \{a_1, a_2 \mid a \in A\}$ and $B' = 2nB + n$. It remains to prove that (A', B') satisfies φ and (A, B) is a yes-instance iff (A', B') is a yes-instance. For this, partition the elements of a solution of (A', B') into the elements of form a_1 and a_2. The items $a \in A$ for which a_1 is in the solution correspond to a solution of (A, B) and vice-versa. □

We can transform the instances of SUBSETSUM-φ to PARTITION using the same construction as for Theorem 2.1. Recall that we added two items p and q. In every feasible partition the added items are in different sets of the partition. Thus the constructed instance A' has a property similar to the instances of SUBSETSUM-φ: If a

Table 1: Summary of obtained bounds. Parenthesis around job characteristics denote that the bound holds with and without these. An asterisk (*) after the citation shows that the reduction was modified. The polynomial terms $\|I\|^{\mathrm{O}(1)}$ in the bounds are omitted. A value in the column *Approx.* denotes that the bound also holds for approximate algorithms with a strictly better approximation ratio than the given number.

Problem	Reduced from	Source	Bound	Approx.
$\langle 1 \mid \mathrm{r}(j), \mathrm{d}(j) \mid \mathrm{any} \rangle$	PARTITION	[10]	$2^{\mathrm{o}(n)}$	
$\langle 1 \mid \mathrm{r}(j) \mid \sum w_j C_j \rangle$	SUBSETSUM	[29]	$2^{\mathrm{o}(n)}$	
$\langle \mathrm{P2} \mid \mid \sum w_j C_j \rangle$	PARTITION	[21]	$2^{\mathrm{o}(n)}$	
$\langle \mathrm{P} \mid \mathrm{prec}, \mathrm{t}(j) = 1 \mid C_{\max} \rangle$	CLIQUE	[20]	$2^{\mathrm{o}(\sqrt{n})}$	$3/2$
$\langle \mathrm{R} \mid \mathrm{t}(j,k) \in \{\mathrm{t}(j), \infty\}, (\mathrm{pmtn}) \mid C_{\max} \rangle$	3-SAT	[6]	$2^{\mathrm{o}(n)}$	$3/2$
$\langle \mathrm{P2} \mid \mathrm{para}, (\mathrm{pmtn}) \mid C_{\max} \rangle$	PARTITION	[10]	$2^{\mathrm{o}(n)}$	
$\langle \mathrm{P} \mid \mathrm{para}, (\mathrm{pmtn} \mid \mathrm{migr}) \mid C_{\max} \rangle$	PARTITION	[4]	$2^{\mathrm{o}(n)}$	$3/2$
$\langle \mathrm{P2} \mid \mathrm{mall}, (\mathrm{pmtn}) \mid C_{\max} \rangle$	PARTITION	[10]*	$2^{\mathrm{o}(n)}$	
$\langle \mathrm{P} \mid \mathrm{mall}, (\mathrm{pmtn} \mid \mathrm{migr}) \mid C_{\max} \rangle$	PARTITION	[4]*	$2^{\mathrm{o}(n)}$	$3/2$
$\langle \mathrm{O3} \mid \mid C_{\max} \rangle$	MONOTONE-NAE-SAT	[31]	$2^{\mathrm{o}(n)}$	$5/4$
$\langle \mathrm{O2} \mid \mid \sum w_j C_j \rangle$	4-PARTITION	[1]*	$2^{\mathrm{o}(n)}$	
$\langle \mathrm{O2} \mid \mathrm{pmtn} \mid \sum C_j \rangle$	BALANCEDPARTITION	[5]	$2^{\mathrm{o}(n)}$	
$\langle \mathrm{O} \mid (\mathrm{pmtn}) \mid \sum C_j \rangle$	3-SAT	[13]*	$2^{\mathrm{o}(n)}$	
$\langle \mathrm{F3} \mid (\mathrm{pmtn}) \mid C_{\max} \rangle$	PARTITION	[11]	$2^{\mathrm{o}(n)}$	
$\langle \mathrm{F2} \mid \mid \sum w_j C_j \rangle$	4-PARTITION	[9]	$2^{\mathrm{o}(n)}$	
$\langle \mathrm{F} \mid (\mathrm{pmtn}) \mid \sum C_j \rangle$	3-SAT	[13]*	$2^{\mathrm{o}(n)}$	
$\langle \mathrm{J2} \mid (\mathrm{pmtn}) \mid C_{\max} \rangle$	PARTITION	[11]	$2^{\mathrm{o}(n)}$	

partition $A = A_1 \dot\cup A_2$ with $\mathrm{s}(A_1) = \mathrm{s}(A_2)$ exists, then $|A_1| = |A_2|$, or more formally they fulfill the predicate φ' defined by

$$\varphi'(A) \iff \forall A_1, A_2 \subseteq A\colon \Big(A = A_1 \dot\cup A_2 \wedge \mathrm{s}(A_1) = \mathrm{s}(A_2) \implies |A_1| = |A_2|\Big).$$

Lemma 2.7. *There is no algorithm that solves* PARTITION-φ' *in time* $2^{\mathrm{o}(n)} \times \|A\|^{\mathrm{O}(1)}$, *unless the ETH fails.*

Interestingly, the restriction SUBSETSUM-φ is a special case of the so called SIZEDSUBSETSUM, for which the cardinality of the set to be found is given as part of the instance, and PARTITION-φ' is a special case of BALANCEDPARTITION, for which only partitions $A = A_1 \dot\cup A_2$ are feasible that satisfy $|A_1| = |A_2|$.

Corollary 2.8. *The problems* SIZEDSUBSETSUM *and* BALANCEDPARTITION *cannot be solved in time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$, *unless the ETH fails.*

3 More Scheduling Problems

We conducted a review of existing reductions in the scheduling area. Our findings are summarized in Table 1. We had to modify some of the existing reductions, in particular those starting from 3-PARTITION, for which no strong reduction is known. We have been able to tweak the reduction to 3D-MATCHING by Garey and Johnson [10], and utilized it to obtain a lower bound of $2^{o(n)} \times \|I\|^{O(1)}$ for 4-PARTITION on $4n$ numbers. Most reductions from 3-PARTITION can be altered to start from 4-PARTITION instead. Detailed descriptions of the reductions and our modifications can be found in Appendix C.

4 Exact Solution in $2^{O(n)}$

We now present an algorithmic framework that can optimally solve many scheduling and packing problems in time $2^{O(n)} \times \|I\|^{O(1)}$. The algorithms optimize general classes of objective functions that include the popular choices $C_{\max}$ and $\sum w_j C_j$. Here, a schedule σ is a pair of functions $\sigma_m \colon J \to [m]$, $\sigma_s \colon J \to \mathbb{N}_0$ that assign to each job its machine and starting time, respectively.

4.1 Sequencing on a Constant Number of Machines

We start with an algorithm that can solve problems that involve precedence or exclusion constraints (e.g. for open shop). We require that the objective function is of the form $f(\sigma) = \mathrm{Op}_{j \in J}\, g_j(\sigma_m(j), \sigma_s(j))$, where Op is one of $\sum$, $\min$, and $\max$. Assume that we want to minimize or maximize f and all functions $g_j(k, \cdot)$ are non-decreasing or non-increasing, respectively. Then for any feasible schedule there is an equivalent *compact schedule*, i.e a schedule in which all jobs start as early as possible. Further define Op' to be $\min$ if we minimize f and $\max$ if we maximize f.

Our algorithm is loosely based on the dynamic programming approach of Held and Karp [12] for sequencing jobs on one machine. In contrast to their setting, we must allow idle time, because it may be beneficial (or even required) to wait for a job to finish on another machine. For this, we create a set T containing all possible starting and finishing times of jobs. A small addition allows our algorithm to deal with job-specific release times, which the algorithm by Held and Karp can not handle.

Lemma 4.1 (A proof can be found in Appendix D.1)**.** *We can compute a set T that contains the starting and finishing times of jobs in all compact schedules in time $2^{O(n)} \times \|I\|^{O(1)}$ and $|T| = 2^{O(n)}$.*

The basic idea of the algorithm is to examine possible *outlines* of schedules. Consider a schedule σ for a subset $S \subseteq J$ of jobs. For each machine $k \in [m]$ there is a

job ℓ_k that is scheduled last, unless it has no jobs. The outline of σ is the restriction $\sigma_{|L}$ of σ to the jobs $L(\sigma) = \{\ell_k \mid k \in [m]$, machine k has jobs$\}$. We denote by $\ell(\sigma)$ the job in $L(\sigma)$ that starts last with respect to σ (ties may be broken arbitrarily). An S-outline is a schedule τ for $L \subseteq S$ that is its own outline such that the placement of $\ell(\tau)$ is feasible and S contains no successor of $\ell(\tau)$. Note that there may be S-outlines that are not the outline of any feasible schedule for S. We denote by $O_S(\tau)$ the set of $(S \setminus \{\ell(\tau)\})$-outlines τ' such that τ' agrees with τ on $L(\sigma) \setminus \{\ell(\tau)\}$, the jobs in $L(\tau') \setminus L(\tau)$ finish before $\tau_{\mathrm{s}}(\ell(\tau))$, and τ' only uses machines that are used by τ.

We use a dynamic program to calculate, for each set $S \subseteq J$ and S-outline τ, the best objective value $\mathrm{B}[S, o]$ of a feasible schedule for S with outline τ. See details in Appendix D.1. This is possible because of the following lemma.

Lemma 4.2 (A proof can be found in Appendix D.1)**.** *Let $S \subseteq J$ be a nonempty set of jobs and τ be an S-outline. Then the following recurrence equation holds:*

$$\mathrm{B}[S, o] = \operatorname*{Op'}_{\tau' \in O_S(\tau)} \mathrm{Op}\Big\{\mathrm{B}[S \setminus \{\ell(\tau)\}, \tau'], g_{\ell(\tau)}(\sigma_{\mathrm{m}}(\ell(\tau)), \sigma_{\mathrm{s}}(\ell(\tau)))\Big\}.$$

There are at most $|T|^m \times (|S|+1)^m = 2^{\mathrm{O}(n)}$ S-outlines and 2^n subsets $S \subseteq J$, so our dynamic program runs in $2^{\mathrm{O}(n)}$ iterations. The objective value of an optimal schedule for all jobs then is $\mathrm{Op'}_{\tau\ J\text{-outline}}\, \mathrm{B}[J, \tau]$.

Our algorithm can solve a broad class of problems, including $\langle \mathrm{O}m \mid \mid f \rangle$, $\langle \mathrm{J}m \mid \mid f \rangle$, $\langle \mathrm{F}m \mid \mid f \rangle$, and $\langle \mathrm{R}m \mid \mathrm{prec}, \mathrm{r}(j), \mathrm{d}(j) \mid f \rangle$, in time $2^{\mathrm{O}(n)} \times \|I\|^{\mathrm{O}(1)}$. It can also be extended for parallel and malleable tasks. For $f \in \{C_{\max}, \sum w_j C_j\}$, the problems $\langle \mathrm{O}3 \mid \mid f \rangle$, $\langle \mathrm{J}3 \mid \mid f \rangle$, $\langle \mathrm{F}3 \mid \mid f \rangle$, and $\langle \mathrm{P}2 \mid \mid f \rangle$ cannot be solved asymptotically faster, unless the ETH fails (see Sect. 3).

4.2 Scheduling on an Arbitrary Number of Machines

We now describe an exact algorithm for scheduling on arbitrary many machines. For a schedule σ and $k \in [m]$ we denote by $\mathrm{J}_{\sigma,k}$ the set $\sigma_{\mathrm{m}}^{-1}(k)$ of jobs to be processed on machine k. Furthermore we denote by $\sigma^{(k)}\colon\ \mathrm{J}_{\sigma,k} \to \mathbb{N}_0, j \mapsto \sigma_{\mathrm{t}}(j)$ the schedule on machine k.

The main idea is again to use dynamic programming over subsets of jobs. For each $S \subseteq J$ and $k \in [m]$ we denote by $\mathrm{B}[S, k]$ the best possible objective value when scheduling the jobs S on the first k machines. For each machine k and set S of jobs the algorithm finds and sequences the jobs $S' \subseteq S$ that should be processed on machine k. It does not look back and modify the schedule on the previously filled machines $1, \ldots, k-1$. Thus we demand that there are no constraints on the starting or finishing times of jobs on different machines (e.g. precedence constraints). We must further assume that the objective function of the whole schedule can be calculated iteratively when adding a new machine with jobs to the current schedule,

i.e. the objective function is of the form $f(\sigma) = \mathrm{Op}_{k\in[m]}\, g_k\Big(\mathrm{J}_{\sigma,k}, \sigma^{(k)}\Big)$, where Op is one of $\sum$, $\min$, and $\max$, and the functions g_k can be computed in time $2^{\mathrm{O}(n)} \times \|I\|^{\mathrm{O}(1)}$. If the functions g_k are of the form as in Sect. 4.1 we can also use the algorithm presented there to sequence the jobs on each machine. We use dynamic programming to calculate the values $\mathrm{B}[S,k]$ (see Appendix D.2) by utilizing the recurrence equation

$$\mathrm{B}[k,S] = \begin{cases} g_1(S) & \text{if } k = 1 \\ \mathrm{Op'}_{S'\subseteq S}\, \mathrm{Op}\Big\{\mathrm{B}[k-1, S\setminus S'], g_k\Big(S', \sigma^*_{S',k}\Big)\Big\} & \text{otherwise,} \end{cases}$$

where $\sigma^*_{S',k}\colon S' \to \mathbb{N}_0$ denotes the optimal schedule of S' on machine k. After computing all values the objective value of an optimal schedule can be read from $\mathrm{B}[m,J]$. The dynamic program needs at most $4^n \times m$ iterations.

We have to be careful with the dependence of the running time on m. On identical machines, i.e. $g_1 = \cdots = g_m$ we can assume $m \leq n$, because an optimal schedule uses at most n machines. For different machines (e.g. scheduling on uniform or unrelated machines) this does not work. However, the m functions (or some parameters to distinguish them) then have to be encoded in the input, so we have $\|I\| = \Omega(m)$. Thus, our algorithm has a total running time of $2^{\mathrm{O}(n)} \times \|I\|^{\mathrm{O}(1)}$.

Our algorithm is able to solve the general problem $\langle \mathrm{R} \,|\, r_j, d_j \,|\, f\rangle$. This contains $\langle 1 \,|\, r_j \,|\, \sum w_j C_j\rangle$, $\langle \mathrm{P2} \,|\, | \sum w_j C_j\rangle$, $\langle 1 \,|\, r_j, d_j \,|\, f\rangle$, and $\langle \mathrm{P2} \,|\, | C_{\max}\rangle$ as special cases. In Sect. 3 we have shown that none of them can be solved asymptotically faster under assumption of the ETH. The algorithm can also be adapted to packing problems with multiple containers, e.g. BINPACKING and MULTIPLEKNAPSACK (see Appendix D.3).

5 Approximation Schemes for Knapsack problems

5.1 The Multiple Knapsack Problem

In contrast to the regular knapsack problem instances of MULTIPLEKNAPSACK (MKS) may contain multiple knapsacks with individual capacities.

Theorem 5.1. *There is no approximation scheme for* MULTIPLEKNAPSACK *with running time* $2^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)}$*, unless ETH fails. This bound even holds for* $m = 2$ *knapsacks of equal capacity and when either*

(i) *all items have the same profit, or*

(ii) *the profit of each item equals its size.*

The case of condition (i) is a natural one: by scaling, we can assume that the profit of each item is 1, i.e. we are maximizing the number of packed items. With condition (ii) we maximize the size of the packed items, which is known as the

multiple subset sum problem. The fastest known PTAS for the general case has a running time of $2^{\mathrm{O}\left(\frac{1}{\varepsilon}\log^4\frac{1}{\varepsilon}\right)} + \|I\|^{\mathrm{O}(1)}$ [17].

Also note that both problem restrictions contain PARTITION as special case. Thus the lower bound $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$ applies to exact algorithms. The running time of the algorithm described in Sect. 4.2 matches this bound.

5.1.1 Instances with a Special Profit Structure.

To prove Theorem 5.1 we embed PARTITION into MKS. We then show that an approximation scheme for MKS can be used to decide PARTITION.

Note that for an instance $I = (A, B)$ of MKS, we can regard A as an instance of PARTITION by ignoring the profits. For each set $\mathcal{I}$ of instances of MKS we define $\mathcal{I}_{\mathrm{P}} = \{A \mid (A, B) \in \mathcal{I}\}$ as the set of corresponding instances of PARTITION.

Lemma 5.2. *Let $\mathcal{I}$ be a set of instances of* MKS*, and $\alpha \geq 1$ such that for every instance $I = (A, B) \in \mathcal{I}$ there is a $C \in \mathbb{N}$ with*

(i) *I has $m = 2$ knapsacks of capacity $\frac{1}{2}\,\mathrm{s}(A)$ (note that $\mathrm{s}(A)$ must be even)*

(ii) *$|C| = \|A\|^{\mathrm{O}(1)}$,*

(iii) *$\mathrm{p}(A) \leq n\alpha C$, and*

(iv) *$\mathrm{p}(a) \geq C$ for each item $a \in A$.*

Unless each instance $A \in \mathcal{I}_{\mathrm{P}}$ can be decided in time $2^{\mathrm{o}(n)} \times \|A\|^{\mathrm{O}(1)}$, there is no approximation scheme that approximates all instances $I \in \mathcal{I}$ within $(1+\varepsilon)$ of the optimum in time $2^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)}$.

Proof. Assume there is an approximation scheme P that finds an $(1+\varepsilon)$-approximate solution for every instance $I \in \mathcal{I}$ in time $2^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)}$. Let an arbitrary instance $I = (A, B) \in \mathcal{I}$ be given. First we point out that a packing that packs all items into the two knapsacks exists if and only if A is a yes-instance of PARTITION. Now let $\varepsilon = \frac{1}{n\alpha}$ and solve I approximately using P_ε.

CLAIM 1. The approximate packing successfully packs all items if possible.

Proof of Claim 1. Recall that $\mathrm{p}(A) \leq n\alpha C$, thus $\frac{1}{n\alpha}\,\mathrm{p}(A) \leq C$. If all items can be packed, the packing found by P_ε has profit at least

$$\begin{aligned}\frac{1}{1+\varepsilon} \times \mathrm{OPT}(I) = \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) \times \mathrm{p}(A) &= \mathrm{p}(A) - \frac{1}{n\alpha+1} \times \mathrm{p}(A) \\ &> \mathrm{p}(A) - \frac{1}{n\alpha} \times \mathrm{p}(A) \geq \mathrm{p}(A) - C.\end{aligned}$$

Since the profit of all items is at least C, there is no unpacked item. □ (Claim 1)

Therefore one can decide whether A admits a partition by testing if a $(1+\varepsilon)$-approximate packing packs all items. Because condition (ii) implies $\|I\| = \|A\|^{\mathrm{O}(1)}$, the required running time is $2^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)} = 2^{\mathrm{o}(n)} \times \|A\|^{\mathrm{O}(1)}$. A contradiction, since not all instances in $\mathcal{I}_{\mathrm{P}}$ can be decided in this running time. □ (Lemma 5.2)

We can now prove the first part of Theorem 5.1. Let $\mathcal{I}$ be the set of all instances of MULTIPLEKNAPSACK that satisfy condition (i) and have items of the same profit. Let $I = (A, B) \in \mathcal{I}$ and $p \in \mathbb{N}$ such that the profit $\mathrm{p}(a) = p$ for each item $a \in A$. By scaling we can assume that $p = 1$. Then conditions (ii) to (iv) hold for $C = 1$ and $\alpha = 1$. Furthermore, the set $\mathcal{I}_{\mathrm{P}}$ actually contains every instance of the PARTITION with even $\mathrm{s}(A)$. However, this restriction does not simplify the problem because instances with odd $\mathrm{s}(A)$ must always be no-instances. By Theorem 2.1 we can apply Lemma 5.2 to get the desired result.

5.1.2 The Multiple Subset Sum Problem.

We have to find a set $\mathcal{I}$ of instances of the multiple subset sum problem that satisfies the preconditions of Lemma 5.2. First, we can restrict ourselves to instances that satisfy the knapsack condition (i). Any set $\mathcal{I}$ of such instances is unambiguously determined by $\mathcal{I}_{\mathrm{P}}$. Therefore we only need to give the set $\mathcal{I}_{\mathrm{P}}$ and α. The conditions (ii) to (iv) can be equivalently expressed as: For each instance $A \in \mathcal{I}_{\mathrm{P}}$ there is a $C \in \mathbb{N}$ with

(ii) $|C| = \|A\|^{\mathrm{O}(1)}$,

(iii) $\mathrm{s}(A) \leq n\alpha C$, and

(iv) $\mathrm{s}(a) \geq C$ for each item $a \in A$.

By a linear reduction from PARTITION-φ' (see Sect. 2.4), we will show that there is such a set $\mathcal{I}_{\mathrm{P}}$ and not every instance $A \in \mathcal{I}_{\mathrm{P}}$ can be solved in time $2^{\mathrm{o}(n)} \times \|A\|^{\mathrm{O}(1)}$ if the ETH holds true. For this, transform the instances of PARTITION-φ' such that the sizes of all items are similar, i.e. every instance A fulfills the predicate $\psi(A)$:

$$\psi(A) \iff \exists\, C \in \mathbb{N}\, \forall\, a \in A\colon C \leq \mathrm{s}(a) \leq 3C.$$

Lemma 5.3 (A proof can be found in Appendix E.1)**.** *There is no algorithm that decides* PARTITION-ψ *in time* $2^{\mathrm{o}(n)} \times \|A\|^{\mathrm{O}(1)}$*, unless the ETH fails.*

Proofsketch. Add a suitably large value C to the size of all items. Since a solution contains exactly $\frac{n}{2}$ elements the target B must be increased by $\frac{n}{2}C$. □

We are now able to prove the second part of Theorem 5.1. Let $\mathcal{I}_{\mathrm{P}}$ be the set of instances of PARTITION-ψ for which $\mathrm{s}(A)$ is even. Observe that $\psi(A)$ implies $\mathrm{s}(A) \leq n3C$ for any instance $A \in \mathcal{I}_{\mathrm{P}}$. The set $\mathcal{I} = \{(A, B_A) \mid A \in \mathcal{I}_{\mathrm{P}}\}$ with $B_A =$

$\left(\frac{1}{2}\,\mathrm{s}(A), \frac{1}{2}\,\mathrm{s}(A)\right)$ will therefore satisfy the preconditions of Lemma 5.2 for $\alpha = 3$. Combining Lemma 5.3 with Lemma 5.2 yields the desired result.

5.2 Multi-dimensional Knapsack

Theorem 5.4 (A proof can be found in Appendix E.2)**.** *There cannot exist PTAS for* 2D-KNAPSACK *with running time* $n^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)}$*, unless the ETH fails.*

Proofsketch. Pătraşcu and Williams [28] showed that, under assumption of the ETH, SIZEDSUBSETSUM with n items and solution size k cannot be decided in time $n^{\mathrm{o}(k)}$. Combined with the reduction to 2D-KNAPSACK by Kulik and Shachnai [19] this yields the proposed bound on the running time. □

This bound asymptotically matches the running time $n^{\mathrm{O}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)}$ of known approximation schemes [2].

References

[1] J. O. Achugbue and F. Y. Chin. "Scheduling the open shop to minimize mean flow time". In: *SIAM Journal on Computing* 11.4 (1982), pp. 709–720.

[2] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. "Approximation algorithms for knapsack problems with cardinality constraints". In: *European Journal of Operational Research* 123.2 (2000), pp. 333–345.

[3] J. Chen, X. Huang, I. Kanj, and G. Xia. "On the computational hardness based on linear FPT-reductions". In: *Journal of Combinatorial Optimization* 11 (2 2006), pp. 231–247.

[4] M. Drozdowski. "On The Complexity of Multiprocessor Task Scheduling". In: *Bulletin of the Polish Academy of Sciences. Technical Sciences*. Vol. 43. 3. 1995, pp. 381–392.

[5] J. Du and J. Y.-T. Leung. "Minimizing Mean Flow Time in Two-Machine Open Shops and Flow Shops". In: *Journal of Algorithms* 14.1 (1993), pp. 24–44.

[6] T. Ebenlendr, M. Křćal, and J. Sgall. "Graph balancing: a special case of scheduling unrelated parallel machines". In: *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2008, pp. 483–490.

[7] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[8] F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Springer, 2010.

[9] M. R. Garey, D. S. Johnson, and R. Sethi. "The complexity of flowshop and jobshop scheduling". In: *Mathematical Operations Research* 1 (1976), pp. 117–129.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of* NP*-Completeness*. W. H. Freeman & Co., 1979.

[11] T. Gonzalez and S. Sahni. "Flowshop and jobshop schedules: complexity and approximation". In: *Operations Research* 26.1 (1978), pp. 36–52.

[12] M. Held and R. Karp. "A Dynamic Programming Approach to Sequencing Problems". In: *Journal of the Society for Industrial and Applied Mathematics* 10.1 (1962), pp. 196–210.

[13] H. Hoogeveen, P. Schuurman, and G. J. Woeginger. "Non-approximability results for scheduling problems with minsum criteria". In: *Journal on Computing* 13.2 (2001), pp. 157–168.

[14] E. Horowitz and S. Sahni. "Computing Partitions with Applications to the Knapsack Problem". In: *Journal of the ACM* 21.2 (1974), pp. 277–292.

[15] R. Impagliazzo and R. Paturi. "On the Complexity of k-SAT". In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 367–375.

[16] R. Impagliazzo, R. Paturi, and F. Zane. "Which Problems Have Strongly Exponential Complexity?" In: *Journal of Computer and System Sciences* 63.4 (2001), pp. 512–530.

[17] K. Jansen. "A Fast Approximation Scheme for the Multiple Knapsack Problem". In: *SOFSEM 2012: Theory and Practice of Computer Science*. Lecture Notes in Computer Science 7147. Springer, 2012, pp. 313–324.

[18] K. Jansen, S. Kratsch, D. Marx, and I. Schlotter. "Bin packing with fixed number of bins revisited". In: *Journal of Computer and System Sciences* 79.1 (2013), pp. 39–49.

[19] A. Kulik and H. Shachnai. "There is no EPTAS for two-dimensional knapsack". In: *Information Processing Letters* 110.16 (2010), pp. 707–710.

[20] J. K. Lenstra and A. H. G. Rinnooy Kan. "Complexity of Scheduling under Precedence Constraints". In: *Operations Research* 26.1 (1978), pp. 22–35.

[21] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. "Complexity of Machine Scheduling Problems". In: *Studies in Integer Programming*. Vol. 1. Annals of Discrete Mathematics. Elsevier, 1977, pp. 343–362.

[22] C. Lenté, M. Liedloff, A. Soukhal, and V. T'kindt. "Exponential-time algorithms for scheduling problems". In: 10th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP'11) (Nymburk, Czech Republic, June 19–24, 2011).

[23] D. Lokshtanov, D. Marx, and S. Saurabh. "Lower bounds based on the Exponential Time Hypothesis". In: *Bulletin of the EATCS*. Vol. 105. 2011, pp. 41–72.

[24] D. Marx. "Parameterized complexity and approximation algorithms". In: *The Computer Journal* 51.1 (2008), pp. 60–78.

[25] T. E. O'Neil. "Sub-Exponential Algorithms for 0/1-Knapsack and Bin Packing". In: *Proceedings of the 2011 International Conference on Foundations of Computer Science*. CSREA Press, 2011, pp. 209–214.

[26] T. E. O'Neil and S. Kerlin. "A simple $2^{O(\sqrt{x})}$-algorithm for Partition and Subset Sum". In: *Proceedings of the 2010 International Conference on Foundations of Computer Science*. CSREA Press, 2010, pp. 55–58.

[27] C. H. Papadimitriou and M. Yannakakis. "Optimization, approximation, and complexity classes". In: *Journal of Computer and System Sciences* 43.3 (1991), pp. 425–440.

[28] M. Pătraşcu and R. Williams. "On the possibility of faster SAT algorithms". In: *Proceedings of the twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2010, pp. 1065–1075.

[29] A. H. G. Rinnooy Kan. *Machine scheduling problems: classification, complexity and computations*. Stenfert Kroese, 1976.

[30] I. Wegener. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Trans. from the German by R. J. Pruim. Springer, 2003.

[31] D. P. Williamson, L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevast'janov, and D. B. Shmoys. "Short shop schedules". In: *Operations Research* 45.2 (1997), pp. 288–294.

A Description of Problems

In this section we describe decision problems discussed in this paper. For each problem we give the parameter(s) that are used to measure the running time of algorithms.

Problem 2D-KNAPSACK
Given: A set A of items with with vectorial sizes $\mathrm{s}(a) \in \mathbb{N}^2$ and profits $\mathrm{p}(a) \in \mathbb{N}$, $a \in A$, a knapsack capacity $B \in \mathbb{N}^2$, and a target profit D.
Parameter: Number n of items
Decide: Is there a subset $S \subseteq A$ of items such that the knapsack capacity is not exceeded and the target profit is met, i.e. $\mathrm{s}(S) \leq B$ (addition and comparison is component-wise) and $\mathrm{p}(S) \geq D$?

Problem 3-SAT
Given: A formula in 3-CNF
Parameter: Number n of variables, number m of clauses
Decide: Is there a satisfying truth assignment?

Problem 3-SAT'
Given: A formula in 3-CNF where each variable appears exactly three times and each literal appears at most twice.
Parameter: Number n of variables, number of clauses m
Decide: Is there a satisfying truth assignment?

Problem 3D-MATCHING
Given: Three disjoint sets X, Y and Z with equal cardinality p and a set $T \subseteq X \times Y \times Z$ of q triples.
Parameter: Number p of points, number q of triples
Decide: Is there a set $M \subseteq T$ (the matching) such that every element of $X \cup Y \cup Z$ is contained in exactly one triple in M?

Problem BALANCEDPARTITION
Given: A set A of items with sizes $\mathrm{s}(a) \in \mathbb{N}$, $a \in A$.
Parameter: Number n of items
Decide: Is there a partition $A = A_1 \dot{\cup} A_2$ such that $|A_1| = |A_2|$ and $\mathrm{s}(A_1) = \mathrm{s}(A_2)$?

Problem BINPACKING
Given: A set A of items with sizes $\mathrm{s}(a) \in \mathbb{N}$, $a \in A$, a bin capacity $B \in \mathbb{N}$ and a number m of bins.
Parameter: Number n of items
Decide: Is there an assignment $f\colon I \to [m]$ of items such that the capacity of no bin is exceeded, i.e. $\mathrm{s}(f^{-1}(i)) \leq B$ for all $i \in [m]$?

Problem m-BINPACKING
Given: A set A of items with sizes $\mathrm{s}(a) \in \mathbb{N}$, $a \in A$, and a bin capacity $B \in \mathbb{N}$.
Parameter: Number n of items
Decide: Is there an assignment $f\colon I \to [m]$ of items such that the capacity of no bin is exceeded, i.e. $\mathrm{s}(f^{-1}(i)) \leq B$ for all $i \in [m]$?

Problem KNAPSACK
Given: A set A of items with with sizes $\mathrm{s}(a) \in \mathbb{N}$ and profits $\mathrm{p}(a) \in \mathbb{N}$, $a \in A$, a knapsack capacity $B - in - N$, and a target profit D
Parameter: Number n of items
Decide: Is there a subset $S \subseteq A$ of items such that the knapsack capacity is not exceeded and the target profit is met, i.e. $\mathrm{s}(S) \leq B_i$ for all $i \in [m]$ and $\mathrm{p}(S) \geq D$?

Problem MONOTONE-NAE-3-SAT
Given: A formula in 3-CNF that contains no negated literals.
Parameter: Number n of variables, number m of clauses
Decide: Is there a truth assignment that satisfies at least one but not all literals of each clause?

Problem MULTIPLEKNAPSACK (MKS)
Given: A set A of items with with sizes $\mathrm{s}(a) \in \mathbb{N}$ and profits $\mathrm{p}(a) \in \mathbb{N}$, $a \in A$, a vector $B = (B_1, \ldots, B_m)$ of knapsack capacities, and a target profit D.
Parameter: Number n of items
Decide: Is there a subset $S \subseteq A$ of items and an assignment $f\colon S \to [m]$ such that the capacities are not exceeded and the target profit is met, i.e. $\mathrm{s}(f^{-1}(i)) \leq B_i$ for all $i \in [m]$ and $\mathrm{p}(S) \geq D$?

Problem MULTIPROCESSORSCHEDULING (MPS)
Given: A set J of jobs with processing times $\mathrm{t}(j) \in \mathbb{N}$, $j \in J$, a number m of machines, and a target makespan D.
Parameter: Number n of jobs
Decide: Is there an assignment $\sigma\colon J \to [m]$ of jobs to machines (the schedule), such that the finishing time of the last job (the makespan) is less than D, i.e $C_{\max} = \max_{i \in [m]} \mathrm{t}(\sigma^{-1}(i)) \leq D$? For calculating $C_{\max}$ we here exploit the fact that the jobs on one machine can be processed in an arbitrary order.

Problem PARTITION
Given: A set A of items with sizes $\mathrm{s}(a) \in \mathbb{N}$, $a \in A$.
Parameter: Number n of items
Decide: Is there a partition $A = A_1 \dot\cup A_2$ such that $\mathrm{s}(A_1) = \mathrm{s}(A_2)$?

Problem k-PARTITION

Given: A set A of kn items, a bound $B \in \mathbb{N}$, and item sizes $\mathrm{s}(a) \in \mathbb{N}$ with $\frac{B}{k+1} \leq \mathrm{s}(a) \leq \frac{B}{k-1}$ for each $a \in A$.
Parameter: Number n of items
Decide: Is there a partition $A = A_1 \dot{\cup} \cdots \dot{\cup} A_n$ into n subsets such that for each $i \in [n]$ we have $\mathrm{s}(A_i) = B$? The constraints on the item sizes imply that $|A_i| = k$ for $i \in [n]$.

Problem SIZEDSUBSETSUM
Given: A set A of items with sizes $\mathrm{s}(a) \in \mathbb{N}$, $a \in A$, a size parameter k, and a target value $B \in \mathbb{N}$.
Parameter: Number n of items
Decide: Is there a subset $A^* \subseteq A$ with $|A^*| = k$ and $\mathrm{s}(A^*) = B$?

Problem SUBSETSUM
Given: A set A of items with sizes $\mathrm{s}(a) \in \mathbb{N}$, $a \in A$, and a target value $B \in \mathbb{N}$.
Parameter: Number n of items
Decide: Is there a subset $A^* \subseteq A$ with $\mathrm{s}(A^*) = B$?

B Deferred Proofs of Sect. 2

B.1 Proof of Theorem 2.1

Because it is of fundamental importance for our paper we give a detailed proof of Theorem 2.1.

Since we already gave a description of the reductions due to Wegener [30] in Sect. 2.1, we describe his proofs of correctness. We also argue that his reductions are strong. The example depicted in Fig. 1 can help understanding the reduction to SUBSETSUM.

CLAIM 1. The reduction from 3-SAT to SUBSETSUM is correct, i.e. (A, B) is a yes-instance if and only if φ is satisfiable.

Proof [30]. First, we consider the n least significant positions $1, \ldots, n$ of the constructed numbers. For each position i, there are exactly two numbers with a '1' at that position, namely a_i and b_i. Since we cannot have a carry in these positions, a solution must choose exactly one of the items a_i and b_i. The choice of a_i represents $x_i = \text{TRUE}$ and the choice of b_i represents $x_i = \text{FALSE}$.

We now consider the m most significant positions $n+1, \ldots, n+m$, each of which is associated to a clause. If a clause C_j contains the literal x_i, then there is a '1' at position $n + j$ of a_i, and if C_j contains $\bar{x}_i$, then there is a 1 at position $n + j$ of b_i. There is a '0' in these positions otherwise. If we have a truth assignment and select between the a_i and b_i accordingly, then the sum in position $n + j$ equals the number

a_1	0	1	0	0	1
a_2	1	0	0	1	0
a_3	1	0	1	0	0
b_1	1	0	0	0	1
b_2	0	1	0	1	0
b_3	0	0	1	0	0
c_1	0	1	0	0	0
c_2	1	0	0	0	0
d_1	0	1	0	0	0
d_2	1	0	0	0	0
B	3	3	1	1	1

Figure 1: The instance of SUBSETSUM constructed from $(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$. The subset $\{a_1, a_3, b_2, c_1, c_2, d_2\}$ corresponds to the satisfying truth assignment $x_1 = x_3 =$ TRUE, $x_2 =$ FALSE and its values sum to B.

of TRUE literals in clause C_j. Note that we can also have no carry in these positions: since each clause contains at most three literals, there are at most five '1's at each position, three from literals and two from the dummy elements c_j and d_j.

So φ is satisfiable if and only if there is a subset of the a_i and b_i that sums to 1 in the n least significant positions, and to one of 1, 2, and 3 in each of the m most significant positions. If the sum at position $n + j$ is one of 1, 2, and 3, we can add a subset of elements c_j and d_j such that the sum at that position is 3. This is not possible if C_j is not satisfied. □

CLAIM 2. The reduction is strong, i.e. $|A| = \mathrm{O}(m)$.

Proof. We can assume that $n \leq 3m$: since there are at most $3m$ occurrences of variables in the formula, $n > 3m$ can only hold if there are unused variables that can be removed. We have $|A| = 2n + 2m \leq 8m = \mathrm{O}(m)$. □

Lemma B.1. SUBSETSUM *with parameter n (the number of items) cannot be decided in time $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$, unless the ETH fails.*

Proof. The existence of an algorithm that decides SUBSETSUM in time $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$ implies that 3-SAT with parameter m (the number of clauses) can be decided in time $2^{\mathrm{o}(m)} \times \|\varphi\|^{\mathrm{O}(1)}$. This contradicts the ETH [16]. □

We now discuss the reduction to PARTITION. Again the proof of correctness follows that of Wegener [30].

CLAIM 3. The reduction from SUBSETSUM to PARTITION is correct, i.e. (A, B) is a yes-instance if and only if A' is.

Proof [30]. The total size of all items is $\mathrm{s}(A') = 4\,\mathrm{s}(A)$ and the components of a valid partition have size $2\,\mathrm{s}(A)$ each. Assume there is a partition $A' = A_1 \dot\cup A_2$ with $\mathrm{s}(A_1) = \mathrm{s}(A_2) = 2\,\mathrm{s}(A)$. Without restriction we can assume that $p \in A_1$. Then $\mathrm{s}(A_1 \setminus \{p\}) = 2\,\mathrm{s}(A) - (2\,\mathrm{s}(A) - B) = B$. Since $\mathrm{s}(p) + \mathrm{s}(q) = (2\,\mathrm{s}(A) - B) + (\mathrm{s}(A) + B) = 3\,\mathrm{s}(A) > 2\,\mathrm{s}(A)$ and negative sizes are not allowed, we know that p and q are in different components of a partition, i.e. $q \in A_2$ and thus $A_1 \setminus \{p\} \subseteq A$ is a solution of the subset sum instance.

On the other hand, if $S \subseteq A$ is a subset with $\mathrm{s}(S) = B$, then $A' = (S \cup \{p\}) \dot\cup (A \setminus S \cup \{q\})$ is a valid partition. □

CLAIM 4. The reduction is strong, i.e. $|A'| = \mathrm{O}(|A|)$.

Proof. Because we created two new items we have $|A'| = |A| + 2 = \mathrm{O}(|A|)$. □

Lemma B.2. PARTITION *with parameter n (the number of items) cannot be decided in time $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$, unless the ETH fails.*

Proof. The existence of an algorithm that decides PARTITION in time $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$ implies that SUBSETSUM with parameter n (the number of items) can be decided in time $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$. According to Lemma B.1 this is not possible, unless the ETH fails. □

B.2 Details for Theorem 2.2

We generalize SUBSETSUM to a simple packing problem: Given a bin of capacity B and a set of items we have to fill the bin as much as possible. A further generalization yields the well-known knapsack problem, where one seeks to maximize the profits $\mathrm{p}(a)$, a value independent from the size, of the packed items. We give the ILP formulations of these problems, which only differ in the objective function.

$$\max \sum_{a \in A} x_a\,\mathrm{s}(a) \qquad \sum_{a \in A} x_a\,\mathrm{s}(a) \le B \qquad x_a \in \{0,1\} \quad \text{for all } a \in A$$

$$\max \sum_{a \in A} x_a\,\mathrm{p}(a) \qquad \sum_{a \in A} x_a\,\mathrm{s}(a) \le B \qquad x_a \in \{0,1\} \quad \text{for all } a \in A$$

Interestingly, above instances do not seem to be solvable faster than instances of the general 0-1-INTEGERPROGRAMMING, although they are rather simple (only one constraint, no negative coefficients).

B.3 Proof of Lemma 2.6

Recall the construction for an instance (A, B) of SUBSETSUM. For each item $a \in A$ we construct two items a_1 and a_2 with $\mathrm{s}(a_1) = 2n\,\mathrm{s}(a) + 1$ and $\mathrm{s}(a_2) = 1$. We create a new instance from $A' = \{a_1, a_2 \mid a \in A\}$ and $B' = 2nB + n$.

CLAIM 1. (A', B') is an instance of SUBSETSUM-φ.

Proof. Let (A', B') be a yes-instance. There is $S' \subseteq A'$ with $\mathrm{s}(S') = B'$. By partitioning the elements of S' into those of form a_1 and a_2, $a \in A$, we get

$$2nB + n = B' = \sum_{a \in S'} \mathrm{s}(a) = 2n \sum_{\substack{a \in A \\ a_1 \in S'}} \mathrm{s}(a) + |S'|$$

and thus

$$2n \left(B - \sum_{\substack{a \in A \\ a_1 \in S'}} \mathrm{s}(a) \right) = |S'| - n = 0. \tag{1}$$

Equation (1) holds since $2n$ divides $|S'| - n$ and $0 \leq |S'| \leq |A'| = 2n$. □

CLAIM 2. The reduction is correct, i.e. (A, B) is a yes-instance if and only if (A', B') is.

Proof. Let (A, B) be a yes-instance and $S \subseteq A$ with $\mathrm{s}(S) = B$. Then

$$\sum_{a \in S} \mathrm{s}(a_1) + \sum_{a \in A \setminus S} \mathrm{s}(a_2) = 2nB + |S| + (n - |S|) = 2nB + n = B',$$

hence $\{a_1 \mid a \in S\} \dot\cup \{a_2 \mid a \in A \setminus S\} \subseteq A'$ is a solution for (A', B').

Let now (A', B') be a yes-instance. As in the proof of Claim 1, (1) holds. This also implies $\mathrm{s}(\{a \in A \mid a_1 \in S'\}) = B$, which means that (A, B) is a yes-instance. □

Also, the number of items in the instance (A', B') is linearly bounded in n. The lemma follows from Theorem 2.1.

C Details for Sect. 3

C.1 Approximating on Unrelated Machines

In this section we consider the restricted assignment problem. This is a scheduling problem where each job j has a set $\mathrm{M}(j)$ of feasible machines, and j can only be processed on these machines. The restricted assignment problem is actually a special case of unrelated scheduling, where the processing time of each job j arbitrarily depends on the machine it is processed on.

It was shown by Ebenlendr, Křčal, and Sgall [6] that, unless $\mathsf{P} = \mathsf{NP}$, this problem cannot be approximated within a factor $\alpha < \frac{3}{2}$, even when each job j has at most two feasible machines on which its running time is 1 or 2. In the standard scheduling notation this problem can be denoted by $\langle \mathrm{R} \mid \mathrm{t}(j,i) \in \{\mathrm{t}(j), \infty\}, \mathrm{t}(j) \in \{1, 2\}, |\mathrm{M}(j)| \leq 2 \mid C_{\max} \rangle$. The restrictions are somewhat minimal: If we reduce the number of allowed

processing times or the number of feasible machines further the problem becomes trivial. Under assumption of the ETH we can extend this result to rule out sub-exponential α-approximate algorithms for $\alpha < \frac{3}{2}$.

Theorem C.1. *For $\alpha < \frac{3}{2}$, there is no α-approximate algorithm for $\langle \mathrm{R} \mid \mathrm{t}(j,i) \in \{\mathrm{t}(j), \infty\}, \mathrm{t}(j) \in \{1,2\}, |\mathrm{M}(j)| \leq 2 \mid C_{\max} \rangle$ with running time $2^{\mathrm{o}(n)} \times |I|^{\mathrm{O}(1)}$, unless the ETH fails.*

On the other hand, our algorithm from Sect. 4, can solve the unrestricted variant of unrelated scheduling in time $2^{\mathrm{O}(n)} \times |I|^{\mathrm{O}(1)}$, so this result is tight.

Proof of Theorem C.1. It is sufficient to find a strong reduction from 3-SAT to the problem $\langle \mathrm{R} \mid \mathrm{t}(j,i) \in \{\mathrm{t}(j), \infty\}, \mathrm{t}(j) \in \{1,2\}, |\mathrm{M}(j)| \leq 2 \mid C_{\max} \rangle$ that has a gap of $\frac{3}{2}$, i.e. the scheduling instance admits a makespan of 2 if the formula is satisfiable and has makespan at least 3 otherwise. The reduction by Ebenlendr, Křćal, and Sgall to the equivalent *graph balancing problem* serves this purpose well. We will describe their reduction transferred to the context of unrelated scheduling.

The Construction. The reduction starts from a special case of 3-SAT which we call 3-SAT', where each variable occurs exactly three times and each literal occurs at most twice. Any formula in 3-CNF can be converted to this form [43]: suppose we are given a formula with n variables and m clauses. Let x be a variable that appears k times in the formula. We replace all appearances of x by new, distinct variables $x^{(1)}, \ldots, x^{(k)}$. Next, we enforce that those variables have the same truth value for every satisfying truth assignment with additional clauses $x^{(i)} \vee \bar{x}^{(i+1)}$, $i \in [k-1]$, and $x^{(k)} \vee \bar{x}^{(1)}$. Because there are at most $3n$ occurrences of literals in the original formula, the resulting formula has at most $3n$ variables and $m + 3n$ clauses. This shows that the reduction is linear in $n + m$.

We now transform this problem to the unrelated scheduling problem, see Fig. 2 for a complete example of the construction. Again, we denote by n and m the number of variables and clauses of a formula φ, respectively. For each variable x we construct two *literal machines* m_x and $m_{\bar{x}}$ that correspond to the two literals of x. We also add an *assignment job* j_x with processing time $\mathrm{t}(j_x) = 2$ and $\mathrm{M}(j_x) = \{m_x, m_{\bar{x}}\}$, i.e. j_x can be processed on the two literal machines.

In addition we create jobs and a machine for each clause C. Let $C = (y_{C,1} \vee \cdots \vee y_{C,k})$ (note that $k \leq 3$). Then, add a *clause machine* m_C, and for each literal $y_{C,i}$ appearing in C add a *literal job* $j_{C,i}$ with $\mathrm{t}(j_{C,i}) = 1$ and $\mathrm{M}(j_{C,i}) = \{m_C, m_{y_{C,i}}\}$. Observe that, since each literal can appear twice in the formula, there can be two *distinct* literal jobs corresponding to the same literal. Finally, if $k < 3$ we add a *dummy job* $j_{C,\mathrm{d}}$ with $\mathrm{M}(j_{C,\mathrm{d}}) = \{m_C\}$ and $\mathrm{t}(j_{C,\mathrm{d}}) = 3 - k$.

It is easy to see that the constructed instance of the restricted assignment problem indeed satisfies $|\mathrm{M}(j)| \leq 2$ for each job j. Furthermore, it contains $n + 3m + m$ jobs, so the reduction is strong.

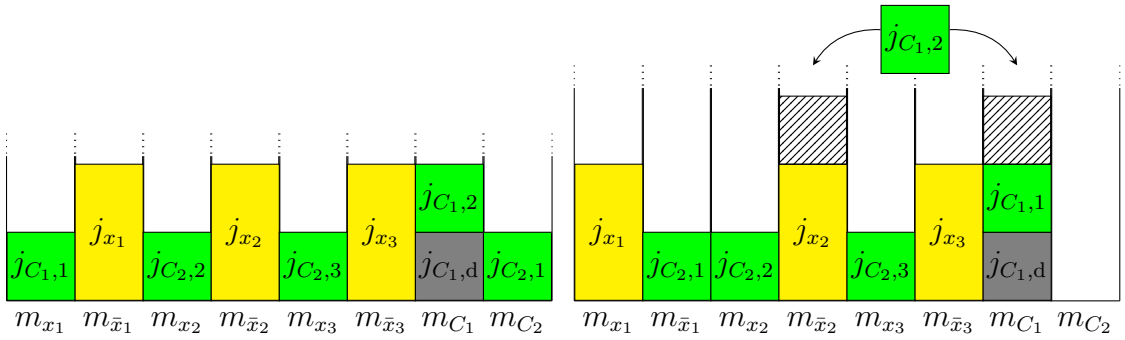


Figure 2: Example of the reduction for the formula $C_1 \wedge C_2 = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$. The first schedule was obtained from the satisfying truth assignment $\beta(x_1) = \beta(x_2) = \beta(x_3) = \text{TRUE}$ and has makespan 2. The second (partial) schedule arises from the non-satisfying assignment $\beta(x_1) = \text{FALSE}$, $\beta(x_2) = \beta(x_3) = \text{TRUE}$ and does not admit an assignment of the job $j_{C_1,2}$ with makespan 2.

The Correctness. We will now argue that the constructed instance admits a schedule with makespan 2 if and only if φ is satisfiable. For a more formal proof we refer to the work of Ebenlendr, Křćal, and Sgall [6].

The idea behind the assignment jobs is that the machines used to process them in a schedule correspond to a truth assignment β: For each variable x, the assignment job j_x is processed on m_x if and only if $\beta(x) = \text{FALSE}$. Otherwise j_x is processed on $m_{\bar{x}}$ and $\beta(x) = \text{TRUE}$. In order to obtain a schedule of makespan 2, we cannot place any literal jobs on the literal machines occupied by the assignment jobs. This means that all literal jobs corresponding to literals that have the truth value FALSE according to β get pushed to their clause machines. Now consider that the formula is not satisfied if and only if at least one clause is not satisfied, and that a clause is not satisfied precisely when all its literals are FALSE. So if a clause is not satisfied by β, all its literal jobs get pushed to the corresponding clause machine, causing a load of 3. The dummy jobs enforce this property for clauses containing less than three literals. See Fig. 2 for an illustration. □

C.2 3D-MATCHING and 4-PARTITION

We now consider the 3-dimensional matching problem. The classical reduction from 3-SAT to 3D-MATCHING due to Garey and Johnson produces instances with $p = |X| = |Y| = |Z| = \Theta(nm)$ and $q = |T| = \Theta(p^2)$, where n and m are the number of variables and clauses in the input formula, respectively [10]. This yields a $2^{o(\sqrt[4]{q})} \times \|I\|^{O(1)}$ bound on the running time of exact algorithms.

We will modify this reduction such that $p = \Theta(m)$ and $q = \Theta(p)$. This implies a better running time bound.

Theorem C.2. *There is no algorithm that decides* 3D-MATCHING *in time* $2^{o(q)} \times \|I\|^{O(1)}$, *unless the ETH fails.*

This bound is tight, since finding a matching is trivially possible in time $2^{O(q)} \times \|I\|^{O(1)}$ by enumerating and testing all subsets $M \subseteq T$.

We describe the original construction [10] and our modifications briefly. Let $C_1, \ldots, C_m$ be the clauses of an arbitrary formula in 3-CNF. We will use a convenient set notation for literals and clauses: let $|C| \leq 3$ be the number of literals in the clause C. Instead of $C = (\ell_1 \vee \cdots \vee \ell_{|C|})$ we write $C = \{\ell_1, \ldots, \ell_{|C|}\}$ and consequently $\ell_1, \ldots, \ell_{|C|} \in C$ as well as $\ell \notin C$ for any literal $\ell \notin \{\ell_1, \ldots, \ell_{|C|}\}$. The constructed instance consists of different types of gadgets, the *variable gadgets*, the *clause gadgets* and the *cleanup gadgets*.

Variable Gadgets. For each variable v a variable gadget is created whose purpose it to model the two possible truth assignments for v: For each clause C there are the four Elements $x_v^C, x_{\bar{v}}^C \in X$, $y_v^C \in Y$ and $z_v^C \in Z$, as well as the two triples $t_{\bar{v}}^C = (x_v^C, y_v^C, z_v^C)$ and $t_v^C = (x_{\bar{v}}^C, y_v^{C'}, z_v^C)$. Here C' is the next clause in a cycle, e.g. if $C = C_i$ then $C' = C_{i+1 \bmod m}$.

There will be no more triples in the other gadgets that contain y_v^C or z_v^C. Therefore any matching must contain either all triples from $T_v = \{t_v^C \mid C \text{ clause}\}$ or all triples from $T_{\bar{v}} = \{t_{\bar{v}}^C \mid C \text{ clause}\}$. We imagine that the choice of T_ℓ, $\ell \in \{v, \bar{v}\}$ reflects the assignment of TRUE or FALSE to v, where selecting the triples from T_ℓ means that ℓ is TRUE. This leaves exactly the elements x_ℓ^C unmatched.

In total, the variable gadgets contain $2nm$ elements from both X and T and nm elements from both Y and Z. Our first modification is a reduction of the size of the variable gadgets: For any clause C for which neither $v \in C$ nor $\bar{v} \in C$ we omit x_v^C, $x_{\bar{v}}^C$, y_v^C, z_v^C, t_v^C and $t_{\bar{v}}^C$. After our modification the variable gadgets contribute $2 \times \sum_{i=1}^m |C_i| \leq 6m$ elements to both X and T and at most $3m$ elements to both Y and Z. Also note that the other gadgets do not contain any elements from X.

Clause Gadgets. There is a clause gadget for each clause $C = \{\ell_1, \ldots, \ell_{|C|}\}$. It contains two elements $y^C \in Y$ and $z^C \in Z$. It also contains triples $(x_{\ell_i}^C, y^C, z^C)$, $i \in [|C|]$. The elements y^C and z^C can only be matched when the triples chosen in the variable gadgets correspond to a truth assignment that satisfies the clause C.

The clause gadgets contribute at most $3m$ triples to T and exactly m elements to both Y and Z.

Cleanup Gadgets. Considering only the elements of the variable and clause gadgets every matching must leave $\frac{1}{2}|X| - m$ elements of X unmatched, even when the formula is satisfiable. Thus there are $\frac{1}{2}|X| - m$ cleanup gadgets. A cleanup gadget is a pair of elements $y \in Y$ and $z \in Z$ that can each match with any element $x_\ell^C \in X$.

The cleanup gadgets contribute $\frac{1}{2}|X| - m$ elements to both Y and Z and $|X| \times (\frac{1}{2}|X| - m)$ triples to T. Our second modification reduces the number of triples in the cleanup gadgets by having a cleanup gadget dedicated to each clause. Let $C = \{\ell_1, \ldots, \ell_{|C|}\}$ be a clause. There are $2\,|C|$ elements of X associated with C (if the modification of the variable gadgets is applied). Exactly half of them can be matched within the variable gadgets and up to one can be matched by the clause gadget. Thus it is sufficient to have $|C| - 1$ pairs of elements $y \in Y$ and $z \in Z$ that can each match with any of the $2\,|C|$ elements $x^C_{\ell_i} \in X$, $i \in [|C|]$. This requires $2\,|C|\,(|C| - 1) \leq 12$ triples. In total the modified cleanup gadgets contain at most $12m$ triples.

With both modifications we get $p = |X| = |Y| = |Z| \leq 6m$ and $q = |T| \leq 21m$ and the theorem follows. An interesting consequence is a lower bound for 4-PARTITION, which can be reduced from 3D-MATCHING [10].

Corollary C.3. *There is no algorithm deciding k-PARTITION for $k \geq 4$ with kn items in time $2^{o(n)} \times \|I\|^{O(1)}$, unless the ETH fails.*

Proof. We describe the reduction due to Garey and Johnson [10]. Consider an instance of 3D-MATCHING consisting of $X = \{x_1, \ldots, x_p\}$, $Y = \{y_1, \ldots, y_p\}$, $Z = \{z_1, \ldots, z_p\}$, and T. We can assume that $q \geq p$, otherwise we can output a trivial no-instance. Create an item for each occurrence of an item in a triple. For $a \in X \cup Y \cup Z$ denote by $n_a \in \mathbb{N}$ the number of triples in which a occurs. We create the items $a^1, \ldots, a^{n_a}$. The sizes of the items are gives as follows:

$$\mathrm{s}\left(a^\ell\right) = \begin{cases} 10r^4 + ir \ \ + 1 & \text{if } a = x_i \text{ and } \ell = 1 \\ 11r^4 + ir \ \ + 1 & \text{if } a = x_i \text{ and } \ell \neq 1 \\ 10r^4 + jr^2 + 2 & \text{if } a = y_j \text{ and } \ell = 1 \\ 11r^4 + jr^2 + 2 & \text{if } a = y_j \text{ and } \ell \neq 1 \\ 10r^4 + kr^3 + 2 & \text{if } a = z_k \text{ and } \ell = 1 \\ \ 8r^4 + kr^3 + 2 & \text{if } a = z_k \text{ and } \ell \neq 1, \end{cases}$$

where $r = 32p$. In addition there is one item t for each triple $t = (x_i, y_j, z_k) \in T$ with $\mathrm{s}(t) = 10r^4 - kr^3 + jr^2 + ir + 8$. The target number B is $40r^4 + 15$.

In total the number of items is $3q + q = 4q$, as there are $3q$ occurrences of items in triples (three for each of the q triples) and q triples. Recall that an instance of 4-PARTITION contains $4n$ items, hence we set $n = q$. The reduction is strong and the statement follows from Theorem C.2. □

C.3 Parallel and Malleable Task Scheduling

C.3.1 Parallel Tasks.

An extension to multiprocessor scheduling are *parallel tasks* [32]. A parallel task j has a value $\mathrm{size}(j) \in [m]$ that denotes how many machines are required simultaneously to process j.

Theorem C.4. *There is no exact algorithm for* $\langle \mathrm{P2} \mid \mathrm{para} \mid C_{\max} \rangle$ *with running time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$*, unless the ETH fails.*

Proof. $\langle \mathrm{P2} \mid \mathrm{para} \mid C_{\max} \rangle$ contains $\langle \mathrm{P2} \mid \mid C_{\max} \rangle$ as a special case where $\mathrm{size}(j) = 1$ for every task j. The theorem follows from Theorem 2.4. □

We can also obtain an inapproximability result from a reduction due to Drozdowski [4]

Theorem C.5. *The problem* $\langle \mathrm{P} \mid \mathrm{para}, \mathrm{t}(j) = 1 \mid C_{\max} \leq 2 \rangle$ *cannot be approximated within any factor* $\alpha < \frac{3}{2}$ *in time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$*, unless the ETH fails.*

Proof. Consider some instance A of PARTITION. Create a job j for each item $a \in A$ with $\mathrm{t}(j) = 1$ and $\mathrm{size}(j) = a$. Then there is a schedule with makespan 2 if and only if A admits a partition. An α-approximate algorithm with $\alpha < \frac{3}{2}$ can distinguish between these two cases. The theorem follows from Theorem 2.1. □

Due to the used reduction the bound only holds if the number m of machines is allowed to be exponential in n. On the other hand there is a PTAS by Jansen and Thöle [37] for the problem if m is bounded by a polynomial in n. We can close the remaining gap: the detailed analysis shows that the running time of the approximation scheme is polynomial in m and n [37]. So if m is sub-exponential in n, i.e. $m = 2^{\mathrm{o}(n)}$, then $m^c = (2^{\mathrm{o}(n)})^c = 2^{\mathrm{o}(n)c} = 2^{\mathrm{o}(n)}$ for any constant c. Therefore the approximation scheme has sub-exponential running time.

C.3.2 Malleable Tasks.

A further generalization is the use of *malleable tasks* [34]. A malleable task j can be parallelized arbitrarily: For any number $k \in [m]$ of machines the value $\mathrm{t}(j, k)$ denotes the running time of j when executed on k machines. In a schedule the number of machines has to be fixed once and cannot be changed later[1]. It is usually required that the function $\mathrm{t}(j, \cdot)$ is non-increasing for each job j.

Theorem C.6. *There is no exact algorithm for* $\langle \mathrm{P2} \mid \mathrm{mall} \mid C_{\max} \rangle$ *with running time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$*, unless the ETH fails.*

[1]Some authors call such tasks *moldable*. Malleable tasks in that notion are allowed to migrate to an arbitrary set of machines of possibly different size.

Proof. We can model any instance of $\langle \mathrm{P2} \mid \mathrm{para} \mid C_{\max} \rangle$ by setting $\mathrm{t}(j,k) = \mathrm{t}(j)$ if $k \geq \mathrm{size}(j)$ and $\mathrm{t}(j,k) = \infty$ (or some sufficiently large value) otherwise. The theorem follows from Theorem C.4. □

We have to be more careful with the second reduction: if we encode the instance by listing the running time for each possible number of machines, the length $\|I\|$ of the encoding will be $\Omega(nm)$. The PTAS by Jansen and Thöle can be adapted for the malleable case and, as for parallel tasks, its running time is polynomial in n and m which is also polynomial in the encoding length – thus we have no hope of showing any inapproximability result in that case.

To extend Theorem C.5 to malleable tasks we must use an encoding such that the encoding length remains subexponential in n even if the number of machines is strongly exponential.

Theorem C.7. *The problem* $\langle \mathrm{P} \mid \mathrm{mall} \mid C_{\max} \leq 2 \rangle$ *cannot be approximated within any factor* $\alpha < \frac{3}{2}$ *in time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$ *if a compact encoding of running times is used and the ETH holds.*

Proof. We can exploit the simple structure of the functions $\mathrm{t}(j,\cdot)$ described in the proof of Theorem C.6 (their value only changes once) and use a more compact encoding. One possible encoding with this property would only specify when and to which value the functions change. □

C.3.3 Monotone Tasks.

A further property that is often assumed is *monotony* of all tasks j, i.e. the work function $\mathrm{w}(j,k) = k \times \mathrm{t}(j,k)$ is non-decreasing in k. When $\mathrm{t}(j,\cdot)$ is strictly decreasing and $\mathrm{w}(j,\cdot)$ is strictly increasing for every job j we speak of strictly monotone tasks. While these special cases received some attention in the literature [38, 40, 41, 42] their complexity status has not been explicitly discussed yet.

Theorem C.8. *Even for strictly monotone tasks,* $\langle \mathrm{P2} \mid \mathrm{mall}, \mathrm{monotone} \mid C_{\max} \rangle$ *is weakly* NP*-hard and cannot be solved in time* $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$*, unless the ETH fails.*

Proof. We slightly modify the reduction to $\langle \mathrm{P2} \mid \mathrm{mall} \mid C_{\max} \rangle$ to obtain strict monotony. Let A be an instance of PARTITION. For each item $a \in A$ create a job j_a with $\mathrm{t}(j_a,1) = 4\,\mathrm{s}(a)$ and $\mathrm{t}(j_a,2) = 2\,\mathrm{s}(a) + 1$. Then $\mathrm{t}(j_a,\cdot)$ is strictly decreasing. Furthermore $\mathrm{w}(j_a,1) = \mathrm{t}(j_a,1) = 4\,\mathrm{s}(a)$ and $\mathrm{w}(j_a,2) = 2\,\mathrm{t}(j_a,2) = 4\,\mathrm{s}(a) + 2$, so $\mathrm{w}(j_a,\cdot)$ is strictly increasing. Our jobs are $J = \{j_a \mid a \in A\}$.

CLAIM 3. There is a schedule for $(J,2)$ with makespan $2\,\mathrm{s}(A)$ if and only if A is a yes-instance.

Proof of Claim 3. Let first A be a yes-instance, i.e. there is a partition $A = A_1 \dot\cup A_2$ with $\mathrm{s}(A_1) = \mathrm{s}(A_2) = \frac{1}{2}\,\mathrm{s}(A)$. For each $a \in A$, schedule j_a on machine 1 if $a \in A_1$, and on machine 2 otherwise. The resulting schedule has makespan $2\,\mathrm{s}(A)$.
Now let $\sigma\colon J \to 2^{[2]}$ be a schedule[2] with makespan $2\,\mathrm{s}(A)$.

CLAIM 4. σ schedules each job j on one machine, i.e. $|\sigma(j)| = 1$.

Proof of Claim 4. The *load* of machine k is defined as

$$\mathrm{load}(k) = \sum_{j \in J, k \in \sigma(j)} \mathrm{t}(j, |\sigma(j)|).$$

Assume there is a job j with $|\sigma(j)| > 1$. Then, because $\mathrm{w}(j, \cdot)$ is strictly increasing, we have

$$\sum_{k=1}^{2} \mathrm{load}(k) = \sum_{j \in J} \mathrm{w}(j, |\sigma(j)|) > \sum_{j \in J} \mathrm{w}(j, 1) = 4\,\mathrm{s}(A).$$

By the pigeonhole principle, the load of at least one machine is larger than $2\,\mathrm{s}(A)$, a contradiction. □

From Claim 4 we gather that

$$\sum_{k=1}^{2} \mathrm{load}(k) = \sum_{j \in J} \mathrm{w}(j, 1) = 4\,\mathrm{s}(A),$$

thus $\mathrm{load}(1) = \mathrm{load}(2) = 2\,\mathrm{s}(A)$. Therefore the sets $A_1 = \{a \in A \mid j_a \in \sigma^{-1}(\{1\})\}$ and $A_2 = \{a \in A \mid j_a \in \sigma^{-1}(\{2\})\}$ form the desired partition. □

The reduction shows that, since PARTITION is NP-hard][30], $\langle \mathrm{P2} \mid \mathrm{mall} \mid C_{\max} \rangle$ is also NP-hard. Because the reduction is strong, we get the bound on the running time from Lemma 2.7. □ (Theorem C.8)

It is unknown whether scheduling monotone malleable tasks is strongly NP-hard for any fixed or arbitrary m and whether it admits an approximation ratio better than $\frac{3}{2}$.

C.3.4 Preemptions and Migrations.

All of the above results also hold when preemptions are allowed, as preemptions do not allow improved schedules for the constructed instances.

If migration is also allowed there is an exact algorithm by Jansen and Porkolab [36] whose running time is polynomial in n and m. Thus both parallel and malleable task scheduling are solvable in sub-exponential time $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$ when

[2]For malleable task scheduling, a schedule σ assigns for each job j a *set* $\sigma(j) \subseteq [m]$ of jobs.

m is sub-exponential in n, i.e. $m = 2^{o(n)}$. For exponentially many machines the inapproximability of $\langle \mathrm{P} \,|\, \mathrm{para}, (\mathrm{pmtn}) \,|\, C_{\max} \leq 2\rangle$ and $\langle \mathrm{P} \,|\, \mathrm{mall}, (\mathrm{pmtn}) \,|\, C_{\max} \leq 2\rangle$ within time $2^{o(n)} \times \|I\|^{O(1)}$ still holds.

C.4 Shop Scheduling

In this section we will consider the different shop scheduling variants open shop [35], job shop [33] and flow shop [33], each with and without preemption and with respect to makespan minimization and flow time minimization.

In open shop scheduling there are m different machines. Each job j consists of m *operations* $o_{j,1}, \ldots, o_{j,m}$, and each operation $o_{j,k}$ has to be executed on machine k in time $\mathrm{t}(o_{j,k})$. No two operations of the same job may be executed at the same time. It is allowed that $\mathrm{t}(o_{j,k}) = 0$.

We will show that most shop problems cannot be solved in time $2^{o(n)} \times \|I\|^{O(1)}$. In Sect. 4 we presented an algorithm with running time $2^{O(n)} \times \|I\|^{O(1)}$ for the discussed nonpreemptive problems when the number m of machines is bounded by a constant.

C.4.1 Flow Shop.

In flow shop scheduling the operations $o_{j,1}, \ldots, o_{j,m}$ of each job j have to be processed in exactly this order.

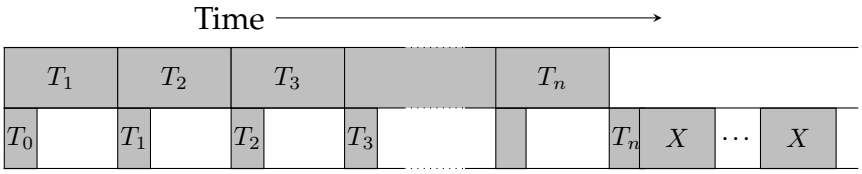
Makespan Minimization. The problems $\langle \mathrm{F2} \,|\, | C_{\max}\rangle$ and $\langle \mathrm{F2} \,|\, \mathrm{pmtn} \,|\, C_{\max}\rangle$ are solvable in polynomial time by the Johnson's algorithm [39]. Gonzalez and Sahni [11] also presented a strong reduction from PARTITION to the problems $\langle \mathrm{F3} \,|\, | C_{\max}\rangle$ and $\langle \mathrm{F3} \,|\, \mathrm{pmtn} \,|\, C_{\max}\rangle$.

Theorem C.9. *The problems* $\langle \mathrm{F3} \,|\, | C_{\max}\rangle$ *and* $\langle \mathrm{F3} \,|\, \mathrm{pmtn} \,|\, C_{\max}\rangle$ *cannot be decided in time* $2^{o(n)} \times \|I\|^{O(1)}$*, unless the ETH fails.*

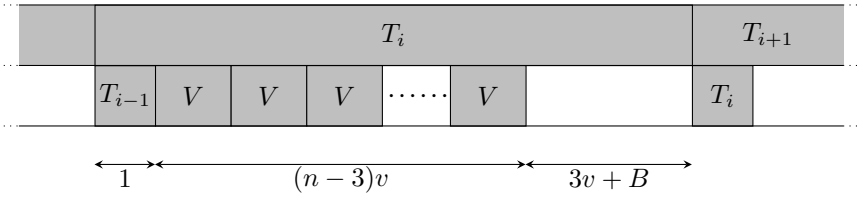
Flow Time Minimization. The problem $\langle \mathrm{F2} \,|\, | \sum C_j\rangle$ is strongly NP-hard by a reduction from 3-PARTITION [9]. The reduction can be easily changed to a strong reduction from 4-PARTITION to $\langle \mathrm{F2} \,|\, | \sum w_j C_j\rangle$.

Theorem C.10. *The problem* $\langle \mathrm{F2} \,|\, | \sum w_j C_j\rangle$ *cannot be decided in time* $2^{o(n)} \times \|I\|^{O(1)}$*, unless the ETH fails.*

Proof. We modify the reduction of Garey, Johnson, and Sethi [9]. Their basic idea is as follows: there are three types of jobs that enforce a certain structure in optimal schedules, n jobs of type T and a super-linear number of types V and X. The jobs of each type are identical except for T_0, which has no operation on machine 1. These jobs should leave n free blocks of $3v + B$ time units each on machine 2 (see Fig. 3).



(a) Overview of the schedule.



(b) Detailed view of one block.

Figure 3: Schedule of the structure jobs.

For each item a of the instance of 3-PARTITION there is a job of type W with only one operation on machine 2 with processing time $v + \mathrm{s}(a)$. The number and length of the jobs is chosen such that no job of type T, X, or V can be delayed even by 1 time unit without significantly increasing the flow time. There is a number D such that the constructed instance has a schedule with flow time $\leq D$ if and only if the instance of 3-PARTITION is a yes-instance.

We have to overcome two problems with this reduction. First, we need to reduce the number of jobs to a linear term. Second, the reduction should start with 4-PARTITION.

Bounding the Number of Jobs Since in every optimal schedule the jobs of type X are scheduled directly after each other, we can replace them with a single job: assume there are n jobs of type X with length l each. When we schedule these directly after each other, starting at time t, they have a total flow time $\sum_{i=1}^{n}(t + il) = nt + l\frac{n(n+1)}{2}$. If we schedule a job with length nl and weight n at time t, it has a weighted flow time of $n(t + nl) = nt + ln^2$. Regardless of the value t the difference remains constant, namely $l\frac{n(n-1)}{2}$. We can subtract this difference from D, and the reduction remains correct. The same modification be applied to replace the jobs of type V with n large jobs. Then the number of jobs is $6n + 2$.

Reducing from 4-PARTITION Because we are reducing from 4-PARTITION instead of 3-PARTITION, we have to fit 4 jobs into the gaps of size $3v + B$. We give the job corresponding to item a the length $\frac{3}{4}v + \mathrm{s}(a)$ and weight $\frac{3}{4}$. Since the ratio of length and weight remain similar, we do not need to change D further. □

Next, we give a strong reduction from 3-SAT′ (see Appendix C.1) to $\langle \mathrm{F} \mid \mid \sum C_j \rangle$. The idea of the reduction is due to Hoogeveen, Schuurman, and Woeginger [13],

who presented a similar L-reduction from BOUNDEDMAX-2-SAT to $\langle \mathrm{F} \mid \mid \sum C_j \rangle$.

Let A and B be minimization problems. An L-reduction from A, to B is a pair of functions R and S that can be computed in polynomial times and have two additional properties:

(i) There is a constant α such that, for any instance I of A, $R(I)$ is an instance of B with $\mathrm{OPT}(R(I)) \leq \alpha \, \mathrm{OPT}(I)$.

(ii) There is a constant β such that for any feasible solution s of $R(I)$, $S(s)$ is is a feasible solution of I with $|\mathrm{OPT}(I) - c(S(s))| \leq \beta \, |\mathrm{OPT}(R(I)) - c(s)|$, where $c(s)$ and $c(S(s))$ are the costs of s and $S(s)$, respectively.

Hoogeveen, Schuurman, and Woeginger used this reduction to show APX-hardness of $\langle \mathrm{F} \mid \mid \sum C_j \rangle$, which implies that there is no PTAS for this problem, unless P $\neq$ NP. Our reduction however is a normal many-one-reduction.

The Construction. Let φ be an instance of 3-SAT'. Consider some variable x and its two literals x and $\bar{x}$. We create two literal jobs for each literal, say x_1, x_2, $\bar{x}_1$ and $\bar{x}_2$. As each of the literals appears at most twice we will consider each of the jobs to correspond to one occurrence of that literal. In addition we create two assignment machines $m^{\mathrm{a}}_{x,1}$, $m^{\mathrm{a}}_{x,2}$ and two consistency machines $m^{\mathrm{c}}_{x,1}$, $m^{\mathrm{c}}_{x,2}$. For each clause C we create one clause machine m_C. We now describe the operations of the literal jobs. Let ℓ_j, $\ell \in \{x, \bar{x}\}$, $j \in \{1, 2\}$ be a literal job. The literal job ℓ_j has 3 operations with processing time 1: One on the assignment machine $m^{\mathrm{a}}_{x,j}$. The second one is on a consistency machine. If $\ell = x$ then the operation is on machine $m^{\mathrm{c}}_{x,j}$, if $\ell = \bar{x}$ then the operation is on machine $m^{\mathrm{c}}_{x,2-j+1}$. The third operation depends on whether x_j occurs in a clause. If so, let C be that clause and execute the operation on m_C. Otherwise we create an additional dummy machine $m^{\mathrm{d}}_{\ell,j}$ and three dummy jobs with one operation of processing time 1 each, to be processed on that dummy machine. The order of the machines is as follows: First are the assignment machines (in arbitrary order), then the consistency machines, and last the clause machines and dummy machines.

Lemma C.11. *Let m_2 and m_3 be the number of clauses in φ with 2 or 3 literals, respectively. The flow shop instance admits a schedule with flow time $26n + 7m_2 + 12m_3$ if and only if φ is satisfiable.*

Proof. A schedule for the constructed instance is called consistent if for each variable x the machines $m^{\mathrm{a}}_{x,1}$ and $m^{\mathrm{a}}_{x,2}$ process either both x_1 and x_2 in the time interval $[0, 1]$ or both $\bar{x}_1$ and $\bar{x}_2$ in the time interval $[0, 1]$. A consistent schedule models a truth assignment: The literals that are scheduled in the interval $[0, 1]$ are considered to be TRUE. Hoogeveen, Schuurman, and Woeginger proved that for every schedule there is a consistent schedule with less or equal flow time.

Assume that φ is satisfiable. Choose a satisfying truth assignment. Schedule the literal jobs corresponding to TRUE literals on the assignment machines in time $[0, 1]$

and on the consistency machines in time $[1, 2]$. Schedule the literal jobs corresponding to FALSE literals on the assignment machines in time $[1, 2]$ and on the consistency machines in time $[2, 3]$. Schedule the clause machines as follows: choose any TRUE literal in the clause and schedule the corresponding operation in time $[2, 3]$. Schedule the (up to two) other operations on this machine immediately after time 3. Schedule the three dummy operations on each dummy machine in time $[0, 3]$ and the operation of the literal job in time $[3, 4]$.

We calculate the flow time of this schedule. The flow time on each assignment machine (consistency machine) is 3 (5). The flow time of a dummy machine is 10. There are exactly $2n$ assignment machines and consistency machines and n dummy machines. thus the total flow time is $f = 26n + 7m_2 + 12m_3$.

Assume now that φ is not satisfiable. Take an optimal and consistent schedule. The schedule is similar to the one described above, except that the clause machines of unsatisfied clauses have no operation in the time interval $[2, 3]$. Because there must be at least one such machine the total flow time is at least $f + 2$. □

It follows immediately:

Theorem C.12. *The problem* $\langle \mathrm{F} \mid (\mathrm{pmtn}) \mid \sum C_j \rangle$ *cannot be solved in time* $2^{o(n)} \times \|I\|^{O(1)}$, *unless ETH fails.*

Whether $\langle \mathrm{F}m \mid \mid \sum C_j \rangle$ admits a solution in time $2^{o(n)} \times \|I\|^{O(1)}$ for any fixed $m > 1$ remains open.

C.4.2 Open Shop.

In open shop scheduling there are no precedence constraints imposed on the operations.

Makespan Minimization. The problems $\langle \mathrm{O2} \mid \mid C_{\max} \rangle$ and $\langle \mathrm{O} \mid \mathrm{pmtm} \mid C_{\max} \rangle$ are solvable in time polynomial in n by an algorithm due to Gonzalez and Sahni [35]. They also presented a strong reduction from PARTITION to $\langle \mathrm{O3} \mid \mid C_{\max} \rangle$ [35].

Theorem C.13. $\langle \mathrm{O3} \mid \mid C_{\max} \rangle$ *cannot be decided in time* $2^{o(n)} \times \|I\|^{O(1)}$, *unless the ETH fails.*

For the next result we use a strong reduction from MONOTONE-NAE-3-SAT to $\langle \mathrm{O} \mid \mid C_{\max} \rangle$ due to Williamson et al. [31].

Theorem C.14. *For* $\alpha < \frac{5}{4}$ *there is no* α*-approximate algorithm for* $\langle \mathrm{O} \mid \mid C_{\max} \rangle$ *with running time* $2^{o(n)} \times \|I\|^{O(1)}$, *unless the ETH fails.*

We only need to verify the hardness of MONOTONE-NAE-3-SAT.

Lemma C.15. MONOTONE-NAE-3-SAT *cannot be decided in time* $2^{o(n)} \times \|I\|^{O(1)}$*, unless the ETH fails.*

Proof. We describe a strong reduction to MONOTONE-NAE-3-SAT from 3-SAT. Let φ be some formula in 3-CNF with n variables and m clauses. Create for each clause $C = (\ell_1 \vee \ell_2 \vee \ell_3) \in \varphi$ one additional variable x_C, and replace the clause by the two new clauses

$$C' = (\ell_1 \vee \ell_2 \vee x_C) \wedge (\bar{x}_C \vee \ell_3).$$

Call the resulting formula φ'.

CLAIM 5 (Folklore). φ is satisfiable if and only if φ' is NAE-satisfiable (i.e. there is a truth assignment such that each clause of φ' contains a TRUE and a FALSE literal).

Proof of Claim 5. Assume φ is satifiable. Chose any satisfying truth assignment. We have to find a truth assignment for the additional variables. Consider some clause $C = (\ell_1 \vee \ell_2 \vee \ell_3) \in \varphi$.
Case 1: $\ell_1 \vee \ell_2$ is TRUE. Set $x_C =$ FALSE, then C' is NAE-satisfied.
Case 2: $\ell_1 \vee \ell_2$ is FALSE. Then ℓ_3 is TRUE. Set $x_C =$ TRUE to NAE-satisfy C'.
Repeat this procedure for each clause to get an assignment that NAE-satisfies φ'.

Now assume that φ is not satisfiable. We want to show that φ' cannot be NAE-satisfied. Chose any truth assignment for φ'. When restricted to the variables of φ, the truth assignment cannot satisfy φ. Therefore, there must be some clause $C = (\ell_1 \vee \ell_2 \vee \ell_3) \in \varphi$ that is not satisfied, i.e. ℓ_1, ℓ_2, and ℓ_3 are all FALSE. Then $C' \equiv (\text{FALSE} \vee \text{FALSE} \vee x_C) \wedge (\bar{x}_C \vee \text{FALSE})$, hence no assignment for x_C can NAE-satisfy C'. □ (Claim 5)

Now replace each variable x in φ with two new variables y and z. Substitute each occurrence of x by y and each occurrence of $\bar{x}$ by z. Finally, add the clause $y \vee z$ that forces y and z to have different truth values for every NAE-satisfying truth assignment. The resulting formula φ'' contains no negated literals, and φ'' is NAE-satisfiable if and only if φ' is. This concludes our reduction to MONOTONE-NAE-3-SAT.

The resulting formula φ'' has at most $2(n + m)$ variables and $2m + n$ clauses. Therefore we have a strong reduction. The lemma follows. □ (Lemma C.15)

Flow Time Minimization.

Theorem C.16. *The problem* $\langle O2 \mid \mid \sum w_j C_j \rangle$ *cannot be decided in time* $2^{o(n)} \times \|I\|^{O(1)}$*, unless the ETH fails.*

Proof. The problem $\langle O2 \mid \mid \sum C_j \rangle$ is known to be strongly NP-hard by a reduction from 3-PARTITION [1]. The reduction can be changed to a strong reduction from 4-PARTITION to $\langle O2 \mid \mid \sum w_j C_j \rangle$ with the same techniques as in the proof of Theorem C.10. □

The reduction from BOUNDEDMAX-2-SAT to $\langle \mathrm{F} \mid \mid \sum C_j \rangle$ mentioned above can be extended to $\langle \mathrm{O} \mid \mid \sum C_j \rangle$ [13].

Theorem C.17. *$\langle \mathrm{O} \mid (\mathrm{pmtn}) \mid \sum C_j \rangle$ cannot be decided in time $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$, unless the ETH fails.*

Proof. We can apply the same method to our modified reduction from 3-SAT': Let φ be an instance of 3-SAT'. We require the creation of the same jobs and machines as above. In addition there are $18n + 6m$ structure jobs. Each structure job has only one operation of non-zero length, its structure operation.

On each of the $2n$ assignment machines there are three structure operations of length 5. On each of the $2n$ consistency machine there are three structure operations each of length $\frac{1}{3}$ and 5. On each of the m clause machine there are six structure operations of length $\frac{1}{3}$. The idea is that the structure operations of length $\frac{1}{3}$ are scheduled first and the operations with length 5 are scheduled last on their machine in any reasonable schedule.

Similar to Lemma C.11, one can verify that the resulting instance of $\langle \mathrm{O} \mid \mid \sum C_j \rangle$ admits a schedule with flow time $180n+14m_2+19m_3$ if and only if φ is satisfiable. □

It is still open whether $\langle \mathrm{O}m \mid \mid \sum C_j \rangle$ can be solved in time $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$ for any fixed $m > 1$.

With preemptions the situation is clearer. The problem $\langle \mathrm{O2} \mid \mathrm{pmtn} \mid \sum C_j \rangle$ can be linearly reduced from BALANCEDPARTITION. Corollary 2.8 implies:

Theorem C.18. *$\langle \mathrm{O2} \mid \mathrm{pmtn} \mid \sum C_j \rangle$ cannot be decided in time $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$, unless the ETH fails.*

C.4.3 Job Shop.

In job shop scheduling the order in which the operations of a job must be processed is specified on a per-job basis. This is obviously a generalization of flow shop scheduling, thus all lower bounds mentioned for flow shop problems also apply for job shop. In addition, there is strong reduction from PARTITION due to Gonzalez and Sahni [11].

Theorem C.19. *The problems $\langle \mathrm{J2} \mid \mid C_{\max} \rangle$ and $\langle \mathrm{J2} \mid \mathrm{pmtn} \mid C_{\max} \rangle$ cannot be decided in time $2^{\mathrm{o}(n)} \times \|I\|^{\mathrm{O}(1)}$, unless the ETH fails.*

D Details for Sect. 4

D.1 Sequencing on a Constant Number of Machines

First, we describe what constraints our algorithm can handle and how different types of scheduling problems can be incorporated with our model.

Extension to Parallel and Malleable Tasks. To support parallel and malleable tasks we need to change the notion of a schedule such that $\sigma_{\mathrm{m}}\colon J \to 2^{[m]}$, i.e. we assign a set $\sigma_{\mathrm{m}}(j) \subseteq [m]$ of machines to each job. Furthermore, for a set $M \subseteq [m]$ of machines, $\mathrm{t}(j, M)$ is the processing time of job $j \in J$ on M. Other than that no changes in notation or definitions are necessary. This increases the number of S-outlines. Note however, that an outline still cannot contain more than m jobs. Thus there are at most $|T|^m \times (|S|+1)^m \times 2^m \leq n^m 2^{nm^2} \times (n+1)^m = 2^{\mathrm{O}(n)}$ outlines.

Constraints. We now give detailed descriptions of the different constraints that the algorithm can handle.

No overlapping jobs The standard constraint of almost all scheduling and packing problems: The processing of two jobs must not overlap, i.e. if $j_1, j_2 \in J$ are executed on $M_1, M_2 \subseteq [m]$ with $M_1 \cap M_2 \neq \emptyset$, then the processing time intervals of j_1 and j_2 must not intersect.

Infeasible machines/machine sets For any job, the placement on arbitrary machines (or sets of machines for parallel and malleable tasks) may be forbidden.

Release times and deadlines Each job j has a release time $\mathrm{r}(j)$ and a deadline $\mathrm{d}(j)$. It must not begin processing before $\mathrm{r}(j)$ and must complete processing at or before $\mathrm{d}(j)$.

Precedence constraints There is a partial order $\prec$ on J. When $j_1 \prec j_2$ for two jobs $j_1, j_2 \in J$. Then the processing of j_1 must be completed at or before the processing of j_2 begins; j_2 is called a *successor* of j_1.

Exclusion constraints There is a relation $\gtrless$ on J. When $j_1 \gtrless j_2$ for $j_1, j_2 \in J$, the processing of j_1 and j_2 may not intersect, i.e. one must complete at or before the time when the other starts. This type of constraint is required for open shop scheduling.

Shop Scheduling Problems. Our algorithm can solve shop scheduling problems, even when we allow more than m operations per job. A job then may have several operations to be processed on the same machine. In this section, we handle the operations like jobs that can only be scheduled on one machine, thus we consider n to be the number of operations with non-zero processing time. The bounds for $\langle \mathrm{O3} \mid \mid f \rangle$, $\langle \mathrm{J3} \mid \mid f \rangle$, and $\langle \mathrm{F3} \mid \mid f \rangle$ from Sect. 3 also hold with this notation, as we limited the number of operations to m in Sect. 3, so the number of operations is linear in the number of jobs.

We will now prove Lemma 4.1.

Proof of Lemma 4.1. Consider a schedule σ for a set J of jobs and some job $j \in J$ that is processed on machines $M = \sigma_{\mathrm{m}}(j)$. We assume that all jobs start as early as possible, so j either starts at its release time $\mathrm{r}(j)$ and finishes at $\mathrm{r}(j) + \mathrm{t}(j, M)$, or j starts immediately after another job. Then, by induction there is a sequence $j_1, \dots, j_\ell = j$ of jobs that start after each other and j_1 starts at its release time $\mathrm{r}(j_1)$. Note that j_i, $i \in [\ell]$ may be scheduled on a possibly different machine set M_i. The finishing time of j is $\mathrm{r}(j_1) + \sum_{i=1}^{\ell} \mathrm{t}(j_i, M_i)$. So

$$T = \Big\{ \mathrm{r}(j_1) + \sum_{M \subseteq [m]} \mathrm{t}(S_M, M) \;\Big|\; j_1 \in J, S_M \subseteq J \text{ for } M \subseteq [m] \Big\}$$

contains all possible starting and finishing times, where S_M are the jobs on machine set M; and $|T| \leq n \times (2^n)^{2^m} = n \times 2^{n2^m} = 2^{\mathrm{O}(n)}$. □

Before we can prove Lemma 4.2, we need to introduce some more notations. We also elaborate on a few notions that were introduced or used in Sect. 4.1.

Schedules and Feasibility. Let $S \subseteq J$. A *schedule on* S is a pair σ of functions $\sigma_{\mathrm{m}} \colon S \to 2^{[m]}$, $\sigma_{\mathrm{s}} \colon S \to \mathbb{N}_0$ that defines machines and starting times for the jobs in S. Thus a schedule is also a schedule on J. If σ is a schedule on S and $S' \subseteq S$, we define the restriction $\sigma_{|S'} = (\sigma_{\mathrm{m}|S'}, \sigma_{\mathrm{t}|S'})$. Obviously, $\sigma_{|S'}$ is a schedule on S'.

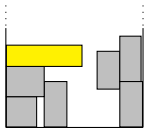
We call a schedule for $S \subseteq J$ feasible, when no constraints are violated that involve only jobs in S. For a job $j \in J$ and a schedule σ we say that the *placement of j is feasible*, if σ does not violate any constraints that involve j. Clearly, a schedule for S is feasible if and only if the placement of each $j \in S$ is feasible.

Evaluation of Schedules on Subsets and $\mathrm{B}[S, \tau]$. Our algorithm works by stepwise extending schedules on subsets of J. We need to extend our objective function f to such schedules: if σ is a feasible schedule for S, then $f(\sigma) = \mathrm{Op}_{j \in S}\, g_j(\sigma_{\mathrm{m}}(j), \sigma_{\mathrm{s}}(j))$. The schedule σ is optimal with outline τ if there is no better schedule for S with outline τ, i.e.

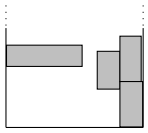
$$f(\sigma) = \mathrm{Op'}\{f(\sigma') \mid \sigma' \text{ is feasible schedule for } S \text{ with outline } \tau\}. \tag{2}$$

The right hand side of (2) is also the definition of the value $\mathrm{B}[S, \tau]$. We use the convention $\mathrm{B}[S, \tau] = \pm\infty$ (depending on whether we minimize or maximize f) to denote that no feasible schedule for S with outline τ exists.

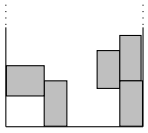
Outlines and S-outlines. The outline τ of a schedule σ is the information that can be observed when looking at σ from above, see Fig. 4. The key observation is that, since $\ell(\sigma)$ is a job that starts last, all jobs of S whose processing time overlaps that of $\ell(\sigma)$ are contained in the set $L(\sigma)$. This inspires the notion of S-outlines: When



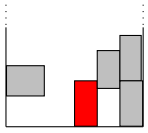
(a) A schedule σ of parallel jobs. The job $\ell(\sigma)$ is highlighted in yellow.



(b) The outline τ of σ.



(c) The outline $\tau' \in O_S(\tau)$ that helps to construct σ.



(d) An outline that is not in $O_S(\tau)$, because the highlighted job uses a machine not used by τ.

Figure 4: Illustration of outlines and related notions.

the placement of $\ell(\tau) = \ell(\sigma)$ is feasible with respect to τ and no successor of $\ell(\tau)$ is contained in S, then the placement of $\ell(\tau)$ is feasible with respect to σ.

Proof of Lemma 4.2. Set $\ell = \ell(\tau)$, $M = \tau_{\mathrm{m}}(\ell) \subseteq [m]$, $t = \tau_{\mathrm{t}}(\ell) \in T$, and $S' = S \setminus \{\ell\}$. Assume that we want to minimize f (the proof for maximization works analog). Here is a restatement of the proposed recurrence equation:

$$\mathrm{B}[S, \tau] = \min_{\tau' \in O_S(\tau)} \mathrm{Op}\{\mathrm{B}[S', \tau'], g_\ell(M, t)\}.$$

Let $\tau' \in O_S(\tau)$ be some outline. Consider an optimal (and thus feasible) schedule σ' for S' with outline τ', i.e. $f(\sigma') = \mathrm{B}[S', \tau']$. We can extend σ' with ℓ to a schedule σ for S with outline τ and σ is feasible. Then

$$\begin{aligned} \mathrm{B}[S, \tau] &\leq f(\sigma) \\ &= \mathrm{Op}\{f(\sigma'), g_\ell(M, t)\} \\ &= \mathrm{Op}\{\mathrm{B}[S', \tau'], g_\ell(M, t)\}. \end{aligned}$$

Since τ' was chosen arbitrarily, we have

$$\mathrm{B}[S, \tau] \leq \min_{\tau' \in O_S(\tau)} \mathrm{Op}\{\mathrm{B}[S', \tau'], g_\ell(M, t)\}.$$

We now show that there is an outline τ' such that equality holds. Consider an optimal schedule σ^* for S with outline τ, i.e. $f(\sigma^*) = \mathrm{B}[S, \tau]$. When we remove the job ℓ we obtain a feasible schedule $\sigma^*_{|S'}$ for S' with an outline τ'. Clearly, $\tau' \in O_S(\tau)$ because $\sigma^*_{|S'}$ is feasible. Now take some optimal schedule σ' for S' with outline τ', i.e. $f(\sigma') = \mathrm{B}[S', \tau']$. Again we can extend σ' to a schedule σ for S with outline τ.

Then we have

$$\begin{aligned}\mathrm{B}[S,\tau] \le f(\sigma) &= \mathrm{Op}\{f(\sigma'), g_\ell(M,t)\}\\ &= \mathrm{Op}\{\mathrm{B}[S',\tau'], g_\ell(M,t)\}\\ &\le \mathrm{Op}\Big\{f\Big(\sigma_{|S'}\Big), g_\ell(M,t)\Big\}\\ &= f(\sigma^*) = \mathrm{B}[S,\tau],\end{aligned}$$

hence we have equality. □

The algorithm. We use a dynamic program to calculate the value $\mathrm{B}[S,\tau]$ for each set $S \subseteq J$ and S-outline τ. The full procedure for minimizing f is given in Algorithm 1. For maximization change ∞ to $-\infty$ in lines 1 and 3 and $<$ to $>$ in lines 2 and 4.

```
   for j ∈ J do # Special handling of |S| = 1
       foreach {j}-outline τ do
           B[{j}, τ] = g_j(τ_m(j), τ_s(j))
   for c = 2 to n do
       foreach S ⊆ J with |S| = c do
           foreach S-outline τ do
1              B[S, τ] = ∞
               for τ' ∈ O_S(τ) do
                   V = Op{B[S', τ'], g_{ℓ(τ)}(τ_m(ℓ(τ)), τ_s(ℓ(τ)))}
2                  if V < B[S, τ] then
                       B[S, τ] = V
                       E[S, τ] = τ'
3  B = ∞
   foreach J-outline τ do # Find optimum
4      if B[J, τ] < B then
           B = B[J, τ]
           τ* = τ
   return B
```

Algorithm 1: Find optimum

In addition, Algorithm 1 tracks some information in the field $\mathrm{E}[S,\tau]$. This is used by Algorithm 2 to construct an optimal schedule.

Analysis of Running Time Let $k = |T|^m(|S|+1)^m 2^m$ a bound on the number of outlines. Algorithm 1 needs at most $nk + 2^n k^2 + k = 2^{\mathrm{O}(n)}$ iterations, and each

$\sigma_{\mathrm{m}} \colon J \to 2^{[m]}$, $\sigma_{\mathrm{t}} \colon J \to \mathbb{N}_0$ # *Initialization: empty schedule*

$\tau = \tau^*$
$S = J$ # *Unscheduled jobs*
while $S \neq \emptyset$ **do**
    $j = \ell(\tau)$
    $\sigma(j) = \tau(j)$
    $\tau = \mathrm{E}[S, \tau]$
    $S = S \setminus \{j\}$
return σ

Algorithm 2: Find schedule

iteration requires time $\|I\|^{\mathrm{O}(1)}$. Thus the total running time is $2^{\mathrm{O}(n)} \times \|I\|^{\mathrm{O}(1)}$. The running time of Algorithm 2 is polynomial in $\|I\|$.

D.2 Scheduling on an Arbitrary Number of Machines

The procedure minimize f is given in Algorithm 3, modifying it for maximization works analog as for Algorithm 1. Algorithm 3 also uses the field $\mathrm{E}[k, S]$ to allow the construction of an optimal schedule, see Algorithm 4 for details.

for $S \subseteq J$ **do** # *Special handling of first machine*
    $\sigma^{(1)} = \texttt{sequence}(S, 1)$
    $\mathrm{B}[1, S] = g_1\Big(S, \sigma^{(1)}\Big)$
    $\mathrm{E}[1, S] = \sigma^{(1)}$
for $k = 2$ **to** m **do**
    for $S \subseteq J$ **do**
        $\mathrm{B}[k, S] = \infty$
        for $S' \subseteq S$ **do**
            $\sigma^{(k)} = \texttt{sequence}(S', k)$
            $V = \mathrm{Op}\Big\{\mathrm{B}[k-1, S \setminus S'], g_k\Big(S', \sigma^{(k)}\Big)\Big\}$
            if $V < \mathrm{B}[k, S]$ **then**
                $\mathrm{B}[k, S] = V$
                $\mathrm{E}[k, S] = \sigma^{(k)}$
return $\mathrm{B}[m, J]$

Algorithm 3: Find optimum

D.3 Packing Problems

Packing problems with multiple containers can also be handled by the algorithm from Sect. 4.2. For bin packing problems we can minimize $f(\sigma) = \sum_{k \in [m]} g_k(\mathrm{J}_{\sigma,k}, \sigma^{(k)})$

$\sigma_{\mathrm{m}} \colon J \to [m]$, $\sigma_{\mathrm{t}} \colon J \to \mathbb{N}_0$ # *Initialization: empty schedule*
$S = J$ # *Unscheduled jobs*
for $k = m$ **to** 1 **do**
  $\sigma^{(k)} = \mathrm{E}[k, S]$
  foreach $j \in \mathrm{dom}(\sigma^{(k)})$ **do**
    $\sigma_{\mathrm{m}}(j) = k$
    $\sigma_{\mathrm{t}}(j) = \sigma^{(k)}(j)$
  $S = S \setminus S'$
return $(\sigma_{\mathrm{m}}, \sigma_{\mathrm{t}})$

Algorithm 4: Find schedule

with

$$g_k(S, \sigma_k) = \begin{cases} 0 & \text{if } S = \emptyset \\ \infty & \text{if } S \text{ cannot be packed in bin } k \\ c_k & \text{otherwise,} \end{cases}$$

where c_k denotes the cost of using bin k. For knapsack-type problems we need to add an additional container $m + 1$ that holds all unpacked items and maximize $f(\sigma) = \sum_{k \in [m+1]} g_k(\mathrm{J}_{\sigma,k}, \sigma^{(k)})$ with

$$g_k(S, \sigma_k) = \begin{cases} 0 & \text{if } k = m + 1 \\ -\infty & \text{if } S \text{ cannot be packed in knapsack } k \\ \mathrm{p}(S, k) & \text{otherwise,} \end{cases}$$

where $\mathrm{p}(S, k)$ denotes the profit of the items S in knapsack k.

E Deferred Proofs from Sect. 5

E.1 Proof of Lemma 5.3

Let A be an instance of PARTITION-φ' We can assume that $n = |A|$ and $\mathrm{s}(A)$ are even, otherwise we can return some trivial no-instance. Choose $C = \frac{1}{2}\,\mathrm{s}(A)$ and add C to the size of all items, i.e. for each item $a \in A$ we create a new item a' with $\mathrm{s}(a') = \mathrm{s}(a) + C$ and the instance $A' = \{a' \mid a \in A\}$. Now A' is an instance of PARTITION-ψ: Let $a \in A$, then we have $C \leq \mathrm{s}(a) + C = \mathrm{s}(a')$ and $\mathrm{s}(a') = \mathrm{s}(a) + C \leq \mathrm{s}(A) + C = 3C$. The number of items does not increase, i.e. the construction is linear. It remains to show that it is a reduction. Extend the notation such that for each $S \subseteq A$ we have $S' = \{a' \mid a \in S\} \subseteq A'$ and vice-versa.

Now let A be a yes-instance. There is a partition $A = A_1 \,\dot\cup\, A_2$ with $\mathrm{s}(A_1) = \mathrm{s}(A_2) = \frac{1}{2}\,\mathrm{s}(A)$. Because ψ is true, we also get $|A_1| = |A_2| = \frac{n}{2}$. Thus, for $i \in \{1, 2\}$ we

have $\mathrm{s}(A'_i) = \mathrm{s}(A_i) + |A_i|C = \frac{1}{2}(\mathrm{s}(A) + nC) = \frac{1}{2}\,\mathrm{s}(A')$, i.e. A' is a yes-instance.

Let now A' have a partition $A' = A'_1 \dot\cup A'_2$ with $\mathrm{s}(A'_1) = \frac{1}{2}\,\mathrm{s}(A') = \frac{1}{2}(\mathrm{s}(A) + nC)$. Furthermore we have $\mathrm{s}(A'_1) = \mathrm{s}(A_1) + |A'_1|C$. It follows that

$$\tfrac{1}{2}\,\mathrm{s}(A) - \mathrm{s}(A_1) = \Big(|A'_1| - \tfrac{n}{2}\Big)C,$$

So C divides $\frac{1}{2}\,\mathrm{s}(A) - \mathrm{s}(A_1)$. Since $\emptyset \neq A_1 \subsetneq A$ we have $0 < \mathrm{s}(A_1) < \mathrm{s}(A)$, hence $\left|\frac{1}{2}\,\mathrm{s}(A) - \mathrm{s}(A_1)\right| < \frac{1}{2}\,\mathrm{s}(A) = C$ also holds. This implies $\frac{1}{2}\,\mathrm{s}(A) - \mathrm{s}(A_1) = 0$. It follows that $\mathrm{s}(A_1) = \frac{1}{2}\,\mathrm{s}(A)$, so A is a yes-instance.

E.2 Proof of Theorem 5.4

There is a reduction R from SIZEDSUBSETSUM to 2D-KNAPSACK due to Kulik and Shachnai [19] with the following property: For any instance $I = (A, B, k)$ of SIZEDSUBSETSUM we have $\mathrm{OPT}(R(I)) \geq k$ if and only if I is a yes-instance.

Assume there is an approximation scheme for 2D-KNAPSACK with running time $n^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|I\|^{\mathrm{O}(1)}$. Then we can solve the subset sum problem with the following procedure. Let $I = (A, B, k)$ be an instance with n numbers of $\mathrm{O}(k \log n)$ bits each and $k \leq n^{0.99}$. We can assume that $\|B\| = \mathrm{O}(\log k \times k \log n)$, otherwise I is a no-instance. Therefore

$$\|I\| = \mathrm{O}(nk \log n + \log k \times k \log n + \log k) = \mathrm{O}\Big(n^3\Big) = n^{\mathrm{O}(1)}$$

and also $\|R(I)\| = \|I\|^{\mathrm{O}(1)} = n^{\mathrm{O}(1)}$.

Now set $\varepsilon = \frac{1}{k}$ and compute an $(1+\varepsilon)$-approximate solution of $R(I)$ by using the approximation scheme. If $\mathrm{OPT}(R(I)) \geq k$ the approximate solution has a profit of at least

$$\frac{1}{1+\varepsilon} \times \mathrm{OPT}(R(I)) = \frac{1}{\frac{k+1}{k}} \times k = \frac{k}{k+1} \times k > \frac{k^2-1}{k+1} = k - 1.$$

Since the solution must be integral, the profit is at least k. If on the other hand $\mathrm{OPT}(R(I)) < k$, the found solution has profit at most $k - 1$.

Thus we can correctly decide if I is a yes-instance by comparing the profit of the approximate solution to k. The running time for this procedure is bounded by $n^{\mathrm{o}\left(\frac{1}{\varepsilon}\right)} \times \|R(I)\|^{\mathrm{O}(1)} + \|I\|^{\mathrm{O}(1)} = n^{\mathrm{o}(k)} \times n^{\mathrm{O}(1)} = n^{\mathrm{o}(k)}$. It has been proven by Pătraşcu and Williams [28] that this implies an algorithm for 3-SAT with running time $2^{\mathrm{o}(n)}$, a contradiction to the ETH.

Additional References

[32] J. Blazewicz, M. Drabowski, and J. Weglarz. "Scheduling Multiprocessor Tasks to Minimize Schedule Length". In: *IEEE Transactions on Computers* C-35.5 (1986), pp. 389–393.

[33] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of scheduling*. Addison-Wesley, 1967.

[34] J. Du and J. Y.-T. Leung. "Complexity of Scheduling Parallel Task Systems". In: *SIAM Journal on Discrete Mathematics* 2.4 (1989), pp. 473–487.

[35] T. Gonzalez and S. Sahni. "Open shop scheduling to minimize finish time". In: *Journal of the ACM* 23.4 (1976), pp. 665–679.

[36] K. Jansen and L. Porkolab. "Computing optimal preemptive schedules for parallel tasks: linear programming approaches". In: *Mathematical Programming* 95.3 (2003), pp. 617–630.

[37] K. Jansen and R. Thöle. "Approximation Algorithms for Scheduling Parallel Jobs". In: *SIAM Journal on Computing* 39.8 (2010), pp. 3571–3615.

[38] K. Jansen and H. Zhang. "An approximation algorithm for scheduling malleable tasks under general precedence constraints". In: *ACM Transactions on Algorithms* 2.3 (2006), pp. 416–434.

[39] S. M. Johnson. "Optimal two- and three-stage production schedules with setup times included". In: *Naval Research Logistics Quarterly* 1.1 (1954), pp. 61–68.

[40] R. Lepère, D. Trystram, and G. J. Woeginger. "Approximation Algorithms for Scheduling Malleable Tasks Under Precedence Constraints". In: *International Journal of Foundations of Computer Science* 13.4 (2002), pp. 613–627.

[41] G. Mounié, C. Rapine, and D. Trystram. "Efficient Approximation Algorithms for Scheduling Malleable Tasks". In: *Proceedings of the Eleventh annual ACM Symposium on Parallel Algorithms and Architectures*. ACM, 1999, pp. 23–32.

[42] G. Mounié, C. Rapine, and D. Trystram. "A $\frac{3}{2}$-Approximation Algorithm for Scheduling Independent Monotonic Malleable Tasks". In: *SIAM Journal on Computing* 37.2 (2007), pp. 401–412.

[43] C. A. Tovey. "A simplified NP-complete satisfiability problem". In: *Discrete Applied Mathematics* 8.1 (1984), pp. 85–89.