Journal of Scheduling (2023) 26:369–382
https://doi.org/10.1007/s10951-022-00771-5

A multivariate complexity analysis of the material consumption scheduling problem

Matthias Bentert[1] · Robert Bredereck[1,2,3] · Péter Györgyi[4] · Andrzej Kaczmarczyk[1,5] · Rolf Niedermeier[1]

Accepted: 15 December 2022 / Published online: 14 January 2023

Abstract
The NP-hard problem Material Consumption Scheduling and related problems have been thoroughly studied since the 1980's. Roughly speaking, the problem deals with scheduling jobs that consume non-renewable resources—each job has individual resource demands. The goal is to minimize the makespan. We focus on the single-machine case without preemption: from time to time, the resources of the machine are (partially) replenished, thus allowing for meeting a necessary precondition for processing further jobs. We initiate a systematic exploration of the parameterized computational complexity landscape of Material Consumption Scheduling, providing parameterized tractability as well as intractability results. Doing so, we mainly investigate how parameters related to the resource supplies influence the problem's computational complexity. This leads to a deepened understanding of this fundamental scheduling problem.

Keywords Non-renewable resources · Makespan minimization · Parameterized computational complexity · Fine-grained complexity · Exact algorithms

1 Introduction

Consider the following motivating example. Every day, an agent works for a number of clients, all of equal importance. The clients, one-to-one corresponding to jobs, each time request a service having individual processing time and individual consumption of a non-renewable resource; examples for such resources include raw material, energy, and money. The goal is to finish all jobs as early as possible, known as minimizing the makespan in the scheduling literature. Unfortunately, the agent only has a limited initial stock of the resource which is to be supplied (with potentially different amounts) at known points in time during the day. Since the job characteristics (resource consumption, job length) and the resource delivery characteristics (delivery amount, point in time) are known in advance, the objective thus is to find a feasible job schedule minimizing the makespan. Notably, jobs cannot be preempted and only one at a time can be executed. Figure 1 provides a concrete numerical example with six jobs having varying job lengths and resource requirements.

The described problem setting is known as minimizing the makespan on a single machine with non-renewable resources. More specifically, we study the single-machine

An extended abstract of this work appeared in the proceedings of the 35th AAAI Conference on Artificial Intelligence 2021 (Bentert et al., 2021). This full version contains full proof details and additional results concerning multiple non-renewable resources.

✉ Matthias Bentert
matthias.bentert@tu-berlin.de

Robert Bredereck
robert.bredereck@tu-clausthal.de

Péter Györgyi
gyorgyi.peter@sztaki.hu

Andrzej Kaczmarczyk
andrzej.kaczmarczyk@agh.edu.pl

Rolf Niedermeier
rolf.niedermeier@tu-berlin.de

1 Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Berlin, Germany

2 Institut für Informatik, Algorithm Engineering, Humboldt-Universität zu Berlin, Berlin, Germany

3 Institut für Informatik, TU Clausthal, Clausthal-Zellerfeld, Germany

4 Institute for Computer Science and Control, Eötvös Loránd Research Network, Budapest, Hungary

5 AGH University, Kraków, Poland

p_j	1	1	1	2	2	3
a_j	3	1	2	3	2	6

u_ℓ	0	3	5	9
$\tilde{b}_\ell$	3	6	2	6

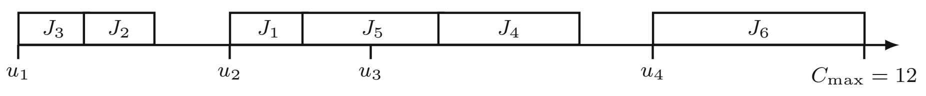

Fig. 1 An example (left) with one resource type and a solution (right) with makespan 12. The processing times and the resource requirements are in the first table, while the supply dates and the supplied quantities are in the second. Note that J_3 and J_2 consume all of the resources supplied at $u_1 = 0$, thus we have to wait for the next supply to schedule further jobs

variant of the NP-hard Material Consumption Scheduling. Formally, we study the following problem.
Material consumption scheduling

Input: A set $\mathcal{R}$ of resources, a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of jobs, each job J_j with a processing time $p_j \in \mathbb{Z}_+$ and a resource requirement $a_{ij} \in \mathbb{Z}_+$ from resource $i \in \mathcal{R}$, a set $\{u_1, u_2, \ldots, u_q\}$ of points in time with $0 = u_1 < u_2 < \cdots < u_q$, and a set $\{\tilde{b}_{i,\ell} \mid i \in R \wedge \ell \in [q]\}$ of resource quantities. At each point u_i in time, $\tilde{b}_{i,\ell}$ quantities of resource $i \in \mathcal{R}$ are supplied.
Task: Find a *schedule* σ with minimum makespan for a single machine without preemption which is *feasible*, that is, (i) the jobs do not overlap in time, and (ii) at any point in time t the total supply from each resource is at least the total request of the jobs starting until t.

The objective is to minimize the makespan, that is, the time at which the last job is completed. Formally, the makespan is defined by $C_{\max} := \max_{J_j \in \mathcal{J}} C_{J_j}$, where C_{J_j} is the completion time of job J_j. Notably, in our example in Fig. 1 we considered the special but perhaps most prominent case of just one type of resource. In this case, we simply drop the indices corresponding to the single resource. In the remainder of the paper, we make the following simplifying assumptions guaranteeing sanity of the instances and filtering out trivial cases.

Assumption 1 Without loss of generality, we assume that

1. for each resource $i \in R$, there are enough resources supplied to process all jobs: $\sum_{\ell=1}^{q} \tilde{b}_{i,\ell} \geq \sum_{J_j \in \mathcal{J}} a_{i,j}$;
2. each job has at least one non-zero resource requirement: $\forall_{J_j \in \mathcal{J}} \sum_{i \in \mathcal{R}} a_{i,j} > 0$; and
3. at least one resource unit is supplied at time 0: $\sum_{i \in \mathcal{R}} \tilde{b}_{i,0} > 0$.

Note that each of these assumptions can be verified in linear time. It is valid to make these assumptions because of the following. If the first assumption does not hold, then there is no feasible schedule. If the second assumption does not hold, then we can schedule all jobs without resource requirements in the beginning. Thus, we can remove these jobs from the instance and adjust the supply times of new resources accordingly. If the third assumption does not hold (but the second does), then we cannot schedule any job before the first resource supply. Thus, we can adjust the supply time of each resource requirement such that there is a resource supply at time 0 and get an equivalent instance.

Material Consumption Scheduling is known to be NP-hard even in the case of just one machine, one resource type, two supply dates ($q = 2$), and if the processing time of each job is the same as its resource requirement, that is, $p_j = a_j$ for each $J_j \in \mathcal{J}$ (Carlier, 1984). While many variants of the problem Material Consumption Scheduling have been studied in the literature in terms of heuristics, polynomial-time approximation algorithms, or the detection of polynomial-time solvable special cases, we are not aware of any previous systematic studies concerning a multivariate complexity analysis. Thus, seemingly for the first time, we study several natural problem-specific parameters and investigate how they influence the computational complexity of the problem. Doing so, we prove both parameterized hardness and fixed-parameter tractability results for this NP-hard problem.

Related Work Over the years, performing multivariate, parameterized complexity studies for fundamental scheduling problems became more and more popular (Bentert et al., 2019; van Bevern et al., 2015; 2017; Bodlaender & Fellows, 1995; Bodlaender & van der Wegen, 2020; Ganian et al., 2020; Fellows & McCartin, 2003; Heeger et al., 2021; Hermelin et al., 2019a; Hermelin et al., 2019b; Hermelin et al., 2020; Hermelin et al., 2019c; Knop & Kouteck'y, 2018; Mnich & van Bevern, 2018; Mnich & Wiese, 2015). We contribute to this field by a seemingly first-time exploration of Material Consumption Scheduling, focusing on one machine and the minimization of the makespan.

Material Consumption Scheduling was introduced in the 1980's (Carlier, 1984; Slowinski, 1984). Indeed, even a bit earlier a problem where jobs required non-renewable resources, but without any machine environment, was studied (Carlier & Rinnooy Kan, 1982). There are several real-world applications, for instance, in the continuous casting stage of steel production (Herr & Goel, 2016), in managing deliveries by large-scale distributors (Belkaid et al., 2012), or in shoe production (Carrera et al., 2010).

Carlier (1984) proved several complexity results for different variants in the single-machine case, while Slowinski (1984) studied the parallel machine variant of the problem with preemptive jobs. Previous theoretical results mainly concentrate on the computational complexity and polynomial-time approximability of different variants; in this literature review, we mainly focus on the most important results for the single-machine case with minimizing makespan as the objective. We remark that there are several recent results for variants with other objective functions (Bérczi et al., 2020; Györgyi & Kis, 2019, 2022), with a more complex machine environment (Györgyi & Kis, 2017), and with slightly different resource constraints (Davari et al., 2020).

Toker et al. (1991) proved that the variant where the jobs require one non-renewable resource reduces to the 2-Machine Flow Shop problem provided that the single non-renewable resource has a unit supply in every time period. Later, Xie (1997) generalized this result to multiple resources. Grigoriev et al. (2005) showed that the variant with unit processing times and two resources is NP-hard. They also provided several polynomial-time 2-approximation algorithms for the general problem. There is also a polynomial-time approximation scheme (PTAS) for the variant with one resource and a constant number of supply dates and a fully polynomial-time approximation scheme (FPTAS) for the case with $q = 2$ supply dates and one non-renewable resource (Györgyi & Kis, 2014). Györgyi and Kis (2015b) presented approximation-preserving reductions between problem variants with $q = 2$ and variants of the Multidimensional Knapsack Problem. These reductions have several consequences; for example, it was shown that the problem is NP-hard if there are two resources, two supply dates, and each job has a unit processing time; or that there is no FPTAS for the problem with two non-renewable resources and $q = 2$ supply dates, unless P = NP. Finally, there are three further results (Györgyi & Kis, 2015a): (i) a PTAS for the variant where the number of resources and the number of supply dates are constants; (ii) a PTAS for the variant with only one resource and an arbitrary number of supply dates if the resource requirements are proportional to job processing times; and (iii) an APX-hardness when the number of resources is part of the input.

Preliminaries and Notation. We employ the standard three-field $\alpha|\beta|\gamma$-notation (Graham et al., 1979), where α denotes the machine environment, β the further constraints like additional resources, and γ the objective function. We always consider a single machine, that is, there is a 1 in the α field. The non-renewable resources are described by nr in the β field and nr $= r$ means that there are r different resource types. In our work, the only considered objective is the makespan $C_{\max}$. The Material Consumption Scheduling variant with a single machine, single resource type, and with the makespan as the objective is then expressed as $1|$ nr $= 1|C_{\max}$. Sometimes, we also consider the so-called *non-idling scheduling* (introduced by Chrétienne 2008), indicated by NI in the α field, in which a machine can only process all jobs continuously, without intermediate idling. As we make the simplifying assumption that the machine has to start processing jobs at time 0, we drop the optimization goal $C_{\max}$ whenever considering non-idling scheduling. When there is just one resource (nr $= 1$), then we write a_j instead of $a_{1,j}$ and $\tilde{b}_j$ instead of $\tilde{b}_{1,j}$, etc. We also write $p_j = 1$ or $p_j = ca_j$ whenever, respectively, jobs have solely unit processing times or the resource requirements are proportional to the job processing times. Finally, we use "unary" to indicate that all numbers in an instance are encoded in unary. Thus, for example, 1, NI $|p_j = 1$, unary $|-$ denotes a single non-idling machine, unit-processing-time jobs and the unary encoding of all numbers. We summarize the notation of the parameters that we consider in the following table.

For simplicity, we introduce the shorthands $a_{\max}$, $\tilde{b}_{\max}$,

n	Number of jobs
q	Number of supply dates
j	Job index
ℓ	Index of a supply
p_j	Processing time of job j
$a_{i,j}$	Resource requirement of job j from resource i
u_ℓ	The ℓ^{th} supply date
$\tilde{b}_{i,\ell}$	Quantity supplied from resource i at u_ℓ
$b_{i,\ell}$	Total resource supply from resource i over the first ℓ supplies, that is, $\sum_{k=1}^{\ell} \tilde{b}_{i,k}$

and $p_{\max}$ for $\max_{J_j \in \mathcal{J}, i \in \mathcal{R}} a_{ij}$, $\max_{\ell \in \{1,\ldots,q\}, i \in \mathcal{R}} \tilde{b}_{i,\ell}$, and $\max_{J_j \in \mathcal{J}} p_j$, respectively. Finally, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

Primer on Multivariate Complexity. To analyze the parameterized complexity (Cygan et al., 2015; Downey & Fellows, 2013; Flum & Grohe, 2006; Niedermeier, 2006) of Material Consumption Scheduling, we define some part of the input as the *parameter* (e.g., the number of supply dates). A parameterized problem is *fixed-parameter tractable* if it is in the class FPT of problems solvable in $f(\rho) \cdot |I|^{O(1)}$ time, where $|I|$ is the size of the given instance encoding, ρ is the value of the parameter, and f is an arbitrary computable (usually super-polynomial) function. Parameterized hardness (and completeness) is defined through parameterized reductions similarly to classical polynomial-time many-one reductions. For our work, it suffices to additionally ensure that the value of the parameter in the problem we reduce to depend only on the value of the parameter of the problem we reduce from. To obtain parameterized intractability, we use parameterized reductions from problems of the class W[1] which is widely believed to be a proper superclass of FPT. For instance, the famous graph problem CLIQUE (Given an undirected graph G and a positive integer k, does G contain k vertices that induce a complete subgraph?) is W[1]-complete with respect to the parameter size k of the clique (Downey & Fellows, 2013).

The class XP contains all problems that are solvable in $|I|^{f(\rho)}$ time for a function f solely depending on the parameter ρ. While XP ensures polynomial-time solvability when ρ is a constant, FPT additionally ensures that the degree of the polynomial is independent of ρ. Unless P = NP, membership in XP can be excluded by showing that the problem is NP-hard for a constant parameter value—for short, we say that the problem is para-NP-hard.

Our Contributions Most of our results are summarized in Table 1. We focus on the parameterized computational complexity of the MATERIAL CONSUMPTION SCHEDULING problem with respect to several parameters describing resource supplies. We show that the case of a single resource and jobs with unit processing time is polynomial-time solvable. However, if each job has a processing time proportional to its resource requirement, then MATERIAL CONSUMPTION SCHEDULING becomes NP-hard even for a single resource and when each supply provides one unit of the resource. Complementing an algorithm solving MATERIAL CONSUMPTION SCHEDULING in polynomial time for a constant number q of supply dates, we show by proving W[1]-hardness that the parameterization by q presumably does not yield fixed-parameter tractability. We circumvent the W[1]-hardness by combining the parameter q with the maximum resource requirement $a_{\max}$ of a job, thereby obtaining fixed-parameter tractability for the combined parameter $q + a_{\max}$. Moreover, we show fixed-parameter tractability for the parameter $u_{\max}$ which denotes the last resource supply time. Finally, we provide an outlook on cases with multiple resources and show that fixed-parameter tractability for $q + a_{\max}$ extends when we additionally add the number r of resources to the combined parameter, that is, we show fixed-parameter tractability for $q + a_{\max} + r$. For MATERIAL CONSUMPTION SCHEDULING with an unbounded number of resources, we show intractability even for the case where all other previously discussed parameters are combined.

2 Computational hardness results

We start our investigation on MATERIAL CONSUMPTION SCHEDULING by outlining the limits of efficient computability. Setting up clear borders of tractability, we identify potential scenarios which are suitable for spotting efficiently solvable special cases. This approach is particularly justified because MATERIAL CONSUMPTION SCHEDULING is already NP-hard for the quite constrained scenario of unit processing times and two resources (Grigoriev et al., 2005).

Both hardness results in this section use reductions from UNARY BIN PACKING. Given a number k of bins, a bin size B, and a set $O = \{o_1, o_2, \ldots, o_n\}$ of n objects of sizes $s_1, s_2, \ldots, s_n$ (encoded in unary), UNARY BIN PACKING asks to distribute the objects to the bins such that no bin exceeds the capacity B. UNARY BIN PACKING is NP-hard and W[1]-hard when parameterized by the number k of bins even if $\sum_{i=1}^{n} s_i = kB$ (Jansen et al., 2013).

We first focus on the case of a single resource, for which we find a strong intractability result. In the following theorem, we show that the problem is NP-hard even if $\tilde{b}_{\max} = 1$.

Theorem 2.1 $1|\,\mathrm{nr} = 1, p_j = ca_j|C_{\max}$ *is para-NP-hard with respect to the maximum number* $\tilde{b}_{\max}$ *of resources supplied at once even if all numbers are encoded in unary.*

Proof Given an initial instance I of UNARY BIN PACKING with $\sum_{i=1}^{n} s_i = kB$, we construct an instance I' of $1|\,\mathrm{nr} = 1, p_j = ca_j|C_{\max}$ with $\tilde{b}_{\max} = 1$ as described below.

We define n jobs $J_1, J_2, \ldots, J_n$ with $J_j = (p_j, a_j)$ such that $p_j = Bs_j$ and $a_j = s_j$. We also introduce a special job $J^* = (p^*, a^*)$, with $p^* = B$ and $a^* = 1$. Then, we set kB supply dates as follows. For each $i \in \{0, 1, \ldots, k-1\}$ and $x \in \{0, 1, \ldots, B-1\}$, we create a supply date $q_i^x = (u_i^x, \tilde{b}_i^x) := ((B + iB^2) - x, 1)$. We add a special supply date $q^* := (0, 1)$. Next, we show that I is a yes-instance if and only if there is a gapless schedule for I', that is, $C_{\max} = kB^2 + B$. An example of this construction is depicted in Fig. 2.

We first show that each solution to I can be transformed to a schedule with $C_{\max} = kB^2 + B$. A yes-instance for I is a partition of the objects into k bins such that each bin is (exactly) full. Formally, there are k sets $S_1, S_2, \ldots S_k$ such that $\bigcup_i S_i = O$, $S_i \cap S_j = \emptyset$ for all $i \neq j$, and $\sum_{o_i \in S_j} s_i = B$ for all j. We form a schedule for I' as follows. First, we schedule job J^* and then, continuously, all jobs corresponding to elements of sets S_1, S_2, and so on. The special supply q^* guarantees that the resource requirement of job J^* is met at time 0. The remaining jobs, corresponding to elements of the partitions, are scheduled earliest at time B, when J^* is processed. The jobs representing each partition, by definition, require in total B resources and take, in total, B^2 time. Thus, it is enough to ensure that at each point $B + iB^2$ in time, for $i \in \{0, 1, \ldots, k-1\}$, there are at least B resources available. This is true because for each $i \in \{0, 1, \ldots, k-1\}$ the time point $B + iB^2$ is preceded with $B - 1$ supplies of one resource. Furthermore, none of the preceding jobs can use the freshly supplied resources as the schedule must be gapless and all processing times are multiples of B. As a result, the schedule is feasible.

Table 1 Our results for a single resource type (top) and multiple resource types (bottom). The results correspond to Theorem 3.1 (‡), Theorem 2.2 (⋄), Györgyi and Kis (2014) (♣), Theorem 2.1 (■), Theorem 3.2 (◊), Theorem 4.1 (▲), Theorem 3.3 (†), Proposition 4.1 (♡), Proposition 4.2 (♠), Theorem 4.2 (▼), and Proposition 4.3 (♦). P stands for polynomial-time solvable, W[1]-h and p-NP stand for W[1]-hardness and para-NP-hardness, respectively

	q	$\tilde{b}_{\max}$	$u_{\max}$	$a_{\max}$	$a_{\max}+q$
$1\|\,\mathrm{nr}=1,\ p_j=1\|C_{\max}$	P‡				
$1\|\,\mathrm{nr}=1,\ p_j=ca_j\|C_{\max}$	W[1]-h$^\diamond$, XP$^\clubsuit$	p-NP$^\blacksquare$	FPT$^\lozenge$	XP$^\blacktriangle$	FPT†
$1\|\,\mathrm{nr}=1,\ \text{unary}\ \|C_{\max}$	W[1]-h$^\diamond$, XP$^\clubsuit$	p-NP$^\blacksquare$	FPT$^\lozenge$	XP$^\blacktriangle$	FPT†
$1\|\,\mathrm{nr}=2,\ p_j=1,\ \text{unary}\ \|C_{\max}$	W[1]-h$^\heartsuit$, XP$^\clubsuit$	p-NP$^\blacksquare$	XP$^\clubsuit$	XP$^\blacktriangle$	FPT$^\spadesuit$
$1\|\,\mathrm{nr}=\text{const},\ \text{unary}\ \|C_{\max}$	W[1]-h$^\heartsuit$, XP$^\clubsuit$	p-NP$^\blacksquare$	XP$^\clubsuit$	XP$^\blacktriangle$	FPT$^\spadesuit$
$1\|\,\mathrm{nr},\ p_j=1\|C_{\max}$	p-NP$^\blacktriangledown$	p-NP$^\blacktriangledown$	W[1]-h$^\blacktriangledown$, XP$^\blacklozenge$	p-NP$^\blacktriangledown$	W[1]-h$^\blacktriangledown$

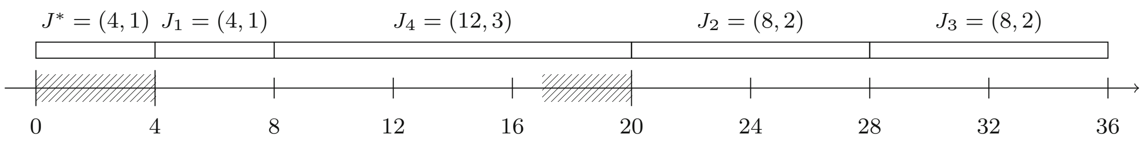

Fig. 2 An example of the construction in the proof of Theorem 2.1 for an instance of UNARY BIN PACKING consisting of $k=2$ bins each of size $B=4$ and four objects o_1 to o_4 of sizes $s_1=1$, $s_2=s_3=2$, and $s_4=3$. In the resulting instance of $1|\,\mathrm{nr}=1,\ p_j=ca_j|C_{\max}$, there are five jobs ($J^*$ and one job corresponding to each input object) and at each (whole) point in time of the hatched periods there is a supply of one resource. An optimal schedule that first schedules J^* is depicted. Note that the time periods between the (right-hand) ends of hatched periods correspond to a multiple of the bin size and a schedule is gapless if and only if the objects corresponding to jobs scheduled between the ends of two consecutive shaded areas exactly fill a bin

Now we show that a gapless schedule for I' implies that I is a yes-instance. Let σ be a gapless schedule for I'. Observe that all processing times are multiples of B and therefore each job has to start at a time that is a multiple of B. For each $i \in \{0,1,\ldots,k-1\}$, we show that there is no job that is scheduled to start before $B+iB^2$ and to end after this time. We show this by induction on i. Since at time 0 there is only one resource available and all jobs that consume one resource have length B, we may assume without loss of generality that job J^* is scheduled first. Hence the statement holds for $i=0$. Assuming that the statement holds for all $i<i'$ for some i', we show that it also holds for i'. Assume toward a contradiction that there is a job J that starts before $t{:=}B+i'B^2$ and ends after this time. Let S be the set of all jobs that were scheduled to start between $t_0{:=}B+(i'-1)B^2$ and t. Recall that for each job $J_{j'}\in S$, we have that $p_{j'}=Ba_{j'}$. Hence, since J ends after t, the number of resources used by S is larger than $(t-t_0)/B=B$. Since only B resources are available at time t, job J cannot be scheduled before time t or there is a gap in the schedule, a contradiction. Hence, there is no job that starts before t and ends after it. Thus, the jobs can be partitioned into "phases," that is, there are $k+1$ sets $S_0, S_1,\ldots,S_k$ such that $S_0=\{J^*\}$, $\bigcup_{h>0} S_h=\mathcal{J}\setminus\{J^*\}$, $S_h\cap S_j=\emptyset$ for all $h\neq j$, and $\sum_{J_j\in S_g} p_j=B^2$ for all g. This corresponds to a bin packing where object o_g belongs to bin $h>0$ if and only if $J_g\in S_h$. □

Note that Theorem 2.1 excludes pseudo-polynomial algorithms for the case under consideration since the theorem statement is true also when all numbers are encoded in unary. Theorem 2.1 motivates to study further problem-specific parameters. Observe that in the reduction presented in the proof of Theorem 2.1, we used an unbounded number of supply dates. Györgyi and Kis (2014) presented a pseudo-polynomial algorithm for $1|\,\mathrm{nr}=1|C_{\max}$ for the case that the number q of supplies is a constant. Thus, the question arises whether we can even obtain fixed-parameter tractability for our problem by taking the number of supply dates as a parameter. Devising a similar (yet slightly simpler) reduction from UNARY BIN PACKING as in the proof of Theorem 2.1, we answer this question negatively in the following theorem.

Theorem 2.2 $1|\,\mathrm{nr}=1,\ p_j=a_j|C_{\max}$ *parameterized by the number q of supply dates is* W[1]*-hard even if all numbers are encoded in unary.*

Proof We reduce from UNARY BIN PACKING parameterized by the number k of bins, which is known to be W[1]-hard (Jansen et al., 2013). Given an instance I of UNARY BIN PACKING with k bins, each of size B and a set O of n objects of sizes $s_1, s_2,\ldots,s_n$ such that $\sum_{i=1}^{n} s_i = kB$, we construct an instance I' of $1|\,\mathrm{nr}=1,\ p_j=a_j|C_{\max}$ as follows.

For each object $o_i\in O$, we define a job $J_i=(p_i,a_i)$ such that $p_i=a_i=s_i$; we denote the set of all jobs by $\mathcal{J}$. We then construct k supply dates $q_i=(u_i,\tilde{b}_i)$, with $u_i=(i-1)B$

and $\tilde{b}_i = B$ for each $i \in [k]$. In this way, we obtain an instance I' of $1|\,\text{nr} = 1|C_{\max}$.

It remains to show that I is a yes-instance if and only if I' is an instance with $C_{\max} = kB$. To this end, suppose first that I is a yes-instance. Then, there is a partition of the objects into k bins such that each bin is (exactly) full. Formally, there are k sets $S_1, S_2, \ldots, S_k$ such that $\bigcup_i S_i = O$, $S_i \cap S_j = \emptyset$ for all $i \neq j$, and $\sum_{o_i \in S_j} s_i = B$ for all j. Hence we schedule all jobs J_i with $o_i \in S_1$ between time $0 = u_1$ and $B = u_2$. Following the same procedure, we schedule all jobs corresponding to objects in S_i between time u_i and u_{i+1} where $u_{i+1} = iB$. Since $\sum_{i=1}^{n} p_i = kB$, we conclude that $C_{\max} = kB$.

Now suppose that $C_{\max} = kB$. Assume toward a contradiction that there is an optimal schedule in which some job $J_i \in \mathcal{J}$ starts at some time t such that $t < u_\ell < t + p_i$ for some $\ell \in [k]$. Let S be the set of all objects that are scheduled before J_i. Since $C_{\max} = kB$, it follows that at each point in time until t there is some job scheduled at this time. Thus, since $p_h = a_h$ for all jobs J_h, it follows that $\sum_{J_h \in S} a_h = \sum_{J_h \in S} p_h = t$ resources are consumed before J_i starts. As a result, $\sum_{J_h \in S \cup \{J_i\}} a_h = t + a_i = t + p_i > u_\ell = (\ell - 1)B$ resources are required to schedule job J_i. Since there are only $\sum_{h=1}^{\ell-1} \tilde{b}_h = (\ell - 1)B$ resources supplied before time $u_\ell > t$, job J_i cannot be scheduled at time t, a contradiction. Hence, there is no job that starts before some u_ℓ and ends after it. Thus, the jobs can be partitioned into k "phases," which means that, there are k sets $S_1, S_2, \ldots, S_k$ such that $\bigcup_h S_h = \mathcal{J}$, $S_i \cap S_{i'} = \emptyset$ for all $i \neq i'$, and $\sum_{J_h \in S_g} p_h = B$ for all g. This corresponds to a bin packing where o_g belongs to bin h if and only if $J_g \in S_h$. □

The theorems presented in this section show that Material Consumption Scheduling is presumably not fixed-parameter tractable with respect to the number of supply dates or with respect to the maximum number of resources per supply. However, we show in the following section that combining these two parameters allows for fixed-parameter tractability. Furthermore, we present other algorithms that, partially, allow us to successfully bypass the hardness presented above.

3 (Parameterized) tractability results

We start our search for efficient algorithms for Material Consumption Scheduling with an introductory part presenting two lemmata exploiting structural properties of problem solutions. Afterwards, we employ the lemmata and provide several tractability results, including polynomial-time solvability for one specific case.

3.1 Identifying structured solutions

A solution to Material Consumption Scheduling is an ordered list of jobs to be executed on the machine(s). Additionally, the jobs need to be assigned their starting times. The starting times have to be chosen in such a way that no job starts when the machine is still processing another scheduled job and that each job requirement is met at the moment of starting the job. We show that, given an ordering of the jobs, one can always compute times of starting the jobs minimizing the makespan in polynomial time. Formally, we present in Lemma 3.1 a polynomial-time Turing reduction from $1|\,\text{nr} = r|C_{\max}$ to $1, \text{NI}\,|\,\text{nr} = r|-$. The crux of this lemma is to observe that there always exists an optimal solution to $1|\,\text{nr} = r|C_{\max}$ that is decomposable into two parts. First, when the machine is idling, and second, when the machine is continuously busy until all jobs are processed. The main idea is the following: Assume that for some instance of Material Consumption Scheduling, there is some optimal schedule where some job J starts being processed at some time t (in particular, the resource requirements of J are met at t). If the machine idles for some time directly after processing job J, then we can postpone processing J to the latest moment which still guarantees that J is ended before the next job is processed. Naturally, at the new starting time of J we can only have more resources than at the old starting time. Applying this observation exhaustively produces a solution that is clearly separated into idling time and busy time.

Lemma 3.1 *There is a polynomial-time Turing reduction from* $1|\,\text{nr} = r|C_{\max}$ *to* $1, \text{NI}\,|\,\text{nr} = r|-$.

Proof Assuming an oracle for $1, \text{NI}\,|\,\text{nr} = r|-$, we describe an algorithm solving $1|\,\text{nr} = r|C_{\max}$ that runs in polynomial time.

We first make a useful observation about feasible solutions to the original problem. Consider some feasible solution σ to $1|\,\text{nr} = r|C_{\max}$ and let $g_1, g_2, \ldots, g_n$ be the idle times before processing, respectively, the jobs $J_1, J_2, \ldots, J_n$. Then, σ can be transformed to another schedule σ' with the same makespan as σ and with idle times $g'_1, g'_2, \ldots, g'_n$ such that $g'_2 = g'_3 \ldots g'_n = 0$ and $g'_1 = \sum_{t \in [n]} g_t$. Intuitively, σ' is a scheduling in which the idle times of σ are all "moved" before the machine starts the first scheduled job. It is straightforward to see that in σ' no jobs are overlapping. Furthermore, each job according to σ' is processed at earliest at the same time as it is processed according to σ. Thus, because there are no "negative" supplies and the order of processed jobs is the same in both σ and σ', each job's resource request is met in schedule σ'.

Using the above observation, the algorithm solving $1|\,\text{nr} = r|C_{\max}$ using an oracle for $1, \text{NI}\,|\,\text{nr} = r|-$ problem works as follows. First, it guesses the duration $g \leq u_{\max}$ of the starting gap and then calls an oracle for $1, \mathrm{NI} \mid \mathrm{nr} = r \mid -$ subtracting g from each supply time (and merging all non-positive supply times to a new one arriving at time zero) of the original $1 \mid \mathrm{nr} = r \mid C_{\max}$ instance. For each value of g, the algorithm adds g to the oracle's output and returns the minimum over all these sums.

Basically, the algorithm finds a schedule with the smallest possible makespan assuming that the idle time happens only before the first scheduled job is processed. Note that this assumption can always be satisfied by the initial observation. Because of the monotonicity of the makespan with respect to the initial idle time g, the algorithm can perform binary search while searching for g and thus its running time is $O(\log(u_{\max}))$. □

We will next extend the idea behind Lemma 3.1 in the subsequent Lemma 3.2 to also change the relative order of certain jobs. We first define a *domination relation* over jobs; intuitively, a job dominates another job if it is not shorter and at the same time its resource consumption is not bigger.

Definition 3.1 A job J_j *dominates* a job $J_{j'}$ (written $J_j \leq_D J_{j'}$) if $p_j \geq p_{j'}$ and $a_{i,j} \leq a_{i,j'}$ for all $i \in \mathcal{R}$.

When we deal with non-idling schedules, for a pair of jobs J_j and $J_{j'}$ where J_j dominates $J_{j'}$, it is better (or at least not worse) to schedule J_j before $J_{j'}$. Indeed, since among these two, J_j's requirements are not greater and its processing time is not smaller, surely after the machine stops processing J_j there will be at least as many resources available as if the machine had processed $J_{j'}$. We formalize this observation in the following lemma. Note that in the case of two jobs J_j and $J_{j'}$ dominating each other ($J_j \leq_D J_{j'}$ and $J_{j'} \leq_D J_j$), we allow for either of them to be processed before the other one.

Lemma 3.2 *For an instance of $1, \mathrm{NI} \mid \mathrm{nr} \mid -$, let $<_D$ be an asymmetric subrelation of $\leq_D$. There always is a feasible schedule where for every pair J_j and $J_{j'}$ of jobs it holds that if $J_j <_D J_{j'}$, then J_j is processed before $J_{j'}$.*

Proof Let σ be some feasible schedule. Consider any pair $(J_j, J_{j'})$ of jobs such that J_j is scheduled after $J_{j'}$ and J_j dominates $J_{j'}$, that is, $p_j \geq p_{j'}$ and $a_{ij} \leq a_{ij'}$ for all $i \in \mathcal{R}$. Denote by σ' a schedule emerging from continuously scheduling all jobs in the same order as in σ but with jobs J_j and $J_{j'}$ swapped. We show that each job in σ' meets its resource requirements, thus proving the lemma. We distinguish between the set of jobs $\mathcal{J}_{\text{out}}$ that are scheduled before $J_{j'}$ in σ or that are scheduled after J_j in σ and jobs $\mathcal{J}_{\text{in}}$ that are scheduled between j' and j in σ (including j' and j). Observe that since all jobs in $\mathcal{J}_{\text{in}}$ are scheduled without the machine idling, it holds that all jobs in $\mathcal{J}_{\text{out}}$ are scheduled exactly at the same times in both σ and σ'. Additionally, since the total number of resources consumed by jobs in $\mathcal{J}_{\text{in}}$ in both σ and σ' is the same, the resource requirements for each job in $\mathcal{J}_{\text{out}}$ is met in σ'. It remains to show that the requirements of all jobs in $\mathcal{J}_{\text{in}}$ are still met after swapping. To this end, observe that all jobs except for $J_{j'}$ still meet the requirements in σ' as J_j dominates $J_{j'}$ (i.e., J_j requires at most as many resources and has at least the same processing time as $J_{j'}$). Thus, each job in $\mathcal{J}_{\text{in}}$ except for $J_{j'}$ has at least as many resources available in σ' as they have in σ. Observe that $J_{j'}$ is scheduled later in σ' than J_j was scheduled in σ. Hence, there are also enough resources available in σ' to process $J_{j'}$. Thus, σ' is feasible, a contradiction. □

3.2 Applying structured solutions

We start with polynomial-time algorithms that apply both Lemma 3.1 and Lemma 3.2 to solve special cases of Material Consumption Scheduling where each two jobs can be compared according to the domination relation (Definition 3.1). Recall that if this is the case, then Lemma 3.2 almost exactly specifies the order in which the jobs should be scheduled.

Theorem 3.1 *$1, \mathrm{NI} \mid \mathrm{nr} \mid -$ and $1 \mid \mathrm{nr} \mid C_{\max}$ are solvable in, respectively, quasilinear and quasiquadratic time if the domination relation is a weak order[1] on the set of jobs. In particular, for the time $u_{\max}$ of the last supply, $1 \mid \mathrm{nr} = 1,\ p_j = 1 \mid C_{\max}$ and $1 \mid \mathrm{nr} = 1,\ a_j = 1 \mid C_{\max}$ are solvable in $O(N \log n \log u_{\max})$ time and $1, \mathrm{NI} \mid \mathrm{nr} = 1,\ p_j = 1 \mid -$ and $1, \mathrm{NI} \mid \mathrm{nr} = 1,\ a_j = 1 \mid -$ are solvable in $O(N \log n)$ time, where N is the size of the input.*

Proof We start with $1, \mathrm{NI} \mid \mathrm{nr} = 1,\ p_j = 1 \mid -$. At the beginning, we order the jobs increasingly with respect to their requirements of the resource, arbitrarily ordering jobs with equal requirements. Then, we simply check whether scheduling the jobs in the computed order yields a feasible schedule, that is, whether the resource requirement of each job is met. If the check fails, then we return "no," otherwise we report "yes." The algorithm is correct due to Lemma 3.2 which, adapted to our case, says that there must exist an optimal schedule in which jobs with smaller resource requirements are always processed before jobs with bigger requirements. It is straightforward to see that the presented algorithm runs in $O(n \log n)$ time.

To extend the algorithm to $1 \mid \mathrm{nr} = 1,\ p_j = 1 \mid C_{\max}$, we apply Lemma 3.1. As described in detail in the proof of Lemma 3.1, we first guess the idling-time g of the machine at the beginning. Then, we run the algorithm for $1, \mathrm{NI} \mid \mathrm{nr} = 1,\ p_j = 1 \mid -$ pretending that we start at time g by shifting backwards by g the times of all resource supplies. Since we

[1] A weak order of elements ranks the elements such that each two objects are comparable but different objects can be tied.

can guess g using binary search in a range from 0 to the time of the last supply, such an adaptation yields a multiplicative factor of $O(\log u_{\max})$ for the running time of the algorithm for $1, \mathrm{NI} \mid \mathrm{nr} = 1,\ p_j = 1|-$. The correctness of the algorithm follows immediately from the proof of Lemma 3.1.

The proofs are analogous for $1, \mathrm{NI} \mid \mathrm{nr} = 1,\ a_j = 1|-$ and $1| \mathrm{nr} = 1,\ a_j = 1|C_{\max}$.

The aforementioned algorithms need only a small modification to work for $1, \mathrm{NI} \mid \mathrm{nr} |-$ and $1| \mathrm{nr} |C_{\max}$. Indeed, we again schedule jobs in a precomputed order. Yet, the respective algorithm uses a general method to compute the weak order of the jobs. First, it sorts descendingly all jobs by their processing times. Then it sorts increasingly all jobs that are so far undistinguishable by resource requirements of an arbitrarily selected resource. The algorithm repeats this procedure iteratively as long as there exist undistinguishable jobs. If, after the last iteration, undistinguishable jobs still exist, then the ties are resolved arbitrarily.[2] Observe that applying this sorting method takes $O(\mathrm{nr} \cdot n \cdot \log n)$ steps, so we get the claimed running time (note that the instance size is at least $\mathrm{nr} \cdot n$). The obtained algorithm is correct by the same argument as the other above-mentioned cases. □

Importantly, it is computationally simple to identify the cases for which the above algorithm can be applied successfully. In fact, the aforementioned procedure for computing the weak order of jobs can, at a computational cost of an additional pass through all jobs once, verify whether the obtained order implies a domination relation.

If the domination relation is not a weak order, then the problem becomes NP-hard as shown in Theorem 2.1. This is to be expected since one cannot efficiently decide which of two incomparable jobs (with respect to the domination relation) to schedule first: the one which requires fewer resource units but has a shorter processing time, or the one that requires more resource units but has a longer processing time. Indeed, it could be the case that sometimes one may want to schedule a shorter job with lower resource consumption to save resources for later, or sometimes it is better to run a long job consuming, for example, all resources knowing that soon there will be another supply with sufficient resource units. Since NP-hardness probably excludes polynomial-time solvability, we turn to a parameterized complexity analysis to get around the intractability.

The time $u_{\max}$ of the last supply seems a promising parameter. We show that it yields fixed-parameter tractability. Intuitively, we demonstrate that the problem is tractable when the time until all resources are available is short.

Theorem 3.2 $1, \mathrm{NI} \mid \mathrm{nr} = 1|-$ *parameterized by the time* $u_{\max}$ *of the last supply is fixed-parameter tractable and can be solved in* $O(2^{u_{\max}} \cdot n + n \log n)$ *time.*

[2] Note that we assume that the domination relation is a weak order.

Proof We first sort all jobs by their processing time in $O(n \log n)$ time. We then sort all jobs with the same processing time by their resource requirements in overall $O(n \log n)$ time. We then iterate over all subsets R of $[u_{\max}]$. We refer to the elements in R as $r_1, r_2, \ldots, r_k$, where $k = |R|$ and $r_i < r_j$ for all $i < j$. For simplicity, we use $r_0 = 0$. For each r_i in ascending order, we check whether there is a job with a processing time $r_i - r_{i-1}$ that was not scheduled before and, if so, then we schedule the respective job with the lowest resource requirement. Next, we check whether there is a job left that can be scheduled at r_k and which has a processing time at least $u_{\max} - r_k$. Finally, we schedule all remaining jobs in an arbitrary order and check whether the total number of resources suffices to run all jobs.

We will now prove that there is a valid gapless schedule if and only if all of these checks are met. Notice that if all checks are met, then our algorithm provides a valid gapless schedule. Now assume that there is a valid gapless schedule. We will show that our algorithm finds a (possibly different) valid gapless schedule. Let, without loss of generality, $J_{j_1}, J_{j_2}, \ldots, J_{j_n}$ be a valid gapless schedule and let j_k be the index of the last job that is processed at time $u_{\max}$. We now focus on the iteration where $R = \{0, p_{j_1}, p_{j_1} + p_{j_2}, \ldots, \sum_{i=1}^{k} p_{j_i}\}$. If the algorithm schedules the jobs $J_{j_1}, J_{j_2}, \ldots, J_{j_k}$, then it computes a valid gapless schedule and all checks are met. Otherwise, it schedules some jobs differently but, by construction, it always schedules a job with processing time p_{j_i} at position $i \le k$. Due to Lemma 3.2, the schedule computed by the algorithm is also valid. Thus, the algorithm computes a valid gapless schedule and all checks are met.

It remains to analyze the running time. The sorting steps in the beginning take $O(n \log n)$ time. There are $2^{u_{\max}}$ iterations for R, each taking $O(n)$ time. Indeed, we can check in constant time for each r_i which job to schedule and this check is done at most n times (as afterwards there is no job left to schedule). Searching for the job that is scheduled at time r_k takes $O(n)$ time as we can iterate over all remaining jobs and check in constant time whether it fulfills both requirements. □

Another way to achieve fixed-parameter tractability via parameters measuring the resource supply structure is combining the parameters q and $\tilde{b}_{\max}$. Although both parameters alone yield intractability, combining them gives fixed-parameter tractability in an almost trivial way: By Assumption 1, every job requires at least one resource, so $\tilde{b}_{\max} \cdot q$ is an upper bound for the number of jobs. Hence, with this parameter combination, we can try out all possible schedules without idling (which by Lemma 3.1 extends to solving $1, \mathrm{NI} \mid \mathrm{nr} = 1|C_{\max}$).

Motivated by this, we replace the parameter $\tilde{b}_{\max}$ by the presumably much smaller (and hence practically more useful) parameter $a_{\max}$. We consider scenarios with only few resource supplies and jobs that require only small units of resources as practically relevant. Next, Theorem 3.3 shows fixed-parameter tractability for the combined parameter $q + a_{\max}$.

Theorem 3.3 $1, \mathrm{NI} \mid \mathrm{nr} = 1 \mid C_{\max}$ *is fixed-parameter tractable for the combined parameter* $q + a_{\max}$*, where* q *is the number of supplies and* $a_{\max}$ *is the maximum resource requirement per job.*

Proof We describe an integer linear program that uses only $f(q, a_{\max})$ variables. To significantly simplify the description of the integer program, we use an extension to integer linear programs that allows concave transformations on variables (Bredereck et al., 2020). While we describe only integer variables used in our program, the extension (Bredereck et al., 2020) allowing concave transformations uses auxiliary fractional variables, thus effectively leading to a mixed integer program (MILP). We then use a famous result of Lenstra (1983), who showed that a mixed integer linear program is fixed-parameter tractable when parameterized by the number of integer variables (see also Frank & Tardos, 1987 and Kannan, 1987 for later asymptotic running-time improvements).

Our approach is based on two main observations. First, by Lemma 3.2 we can assume that there is always an optimal schedule that is consistent with the domination order. Second, within a phase (between two resource supplies), every job can be arbitrarily reordered. Roughly speaking, a solution can be fully characterized by the number of jobs that have been started for each phase and each resource requirement.

We use the following non-negative integer variables:

1. $x_{w,s}$ denoting the number of jobs requiring s resources started in phase w,
2. $x^{\Sigma}_{w,s}$ denoting the number of jobs requiring s resources started in all phases between 1 and w (inclusive),
3. α_w denoting the number of resources available in the beginning of phase w, and
4. d_w denoting the endpoint of phase w, that is, the time when the last job started in phase w ends.

Naturally, the objective is to minimize d_q. First, we ensure that $x^{\Sigma}_{w,s}$ are correctly computed from $x_{w,s}$ by requiring

$$x^{\Sigma}_{w,s} = \sum_{w'=1}^{w} x_{w',s}.$$

Second, we ensure that all jobs are scheduled at some point. To this end, using $\#_s$ to denote the number of jobs J_j with resource requirement $a_j = s$, we add:

$$\forall s \in [a_{\max}]: \quad \sum_{w \in [q]} x_{w,s} = \#_s.$$

Third, we ensure that the α_w variables are set correctly, by setting

$$\alpha_1 = \tilde{b}_1,$$

and

$$\forall 2 \le w \le q: \quad \alpha_w = \alpha_{w-1} + \tilde{b}_w - \sum_{s \in [a_{\max}]} x_{w-1,s} \cdot s.$$

Fourth, we make sure that we always have enough resources:

$$\forall 2 \le w \le q: \quad \alpha_w \ge \tilde{b}_w.$$

Next, we compute the endpoints d_w of each phase, assuming a schedule respecting the domination order. To this end, let p^s_1, $p^s_2, \ldots, p^s_{\#_s}$ denote the processing times of jobs with resource requirement exactly s in non-increasing order. Further, let $\tau_s(y)$ denote the processing time spent to schedule the y longest jobs with resource requirement exactly s, that is, we have $\tau_s(y) = \sum_{i=1}^{y} p^s_i$. Clearly, $\tau_s(x)$ is a concave function that can be precomputed for each $s \in [a_{\max}]$. To compute the endpoints, we add:

$$\forall w \in [q]: \quad d_w = \sum_{s \in [a_{\max}]} \tau_s(x^{\Sigma}_{w,s}). \tag{3.1}$$

Since we assume gapless schedules, we ensure that there is no gap: $\forall 1 \le w \le q-1: d_w \ge u_{w+1} - 1$. This completes the construction of the MILP using concave transformations. The number of integer variables used in the MILP is $2q \cdot a_{\max}$ ($q \cdot a_{\max}$ for $x_{w,s}$ and $x^{\Sigma}_{w,s}$ variables, respectively) plus $2q$ (q for α_w and d_w variables, respectively). Moreover, the only concave transformations used in Constraint Set (3.1) are piecewise linear with only a polynomial number of pieces (in fact, the number of pieces is at most the number of jobs), as required to obtain fixed-parameter tractability of this extended class of MILPs (Bredereck et al., 2020, Theorem 2). □

4 A glimpse on multiple resources

So far we focused on scenarios with only one non-renewable resource. In this section, we provide an outlook on scenarios with multiple resources (still considering only one machine). Naturally, all hardness results transfer. For the tractability results, we identify several cases where tractability extends in some form, while other cases become significantly harder.

Motivated by Theorem 3.3, we are interested in the computational complexity of the MATERIAL CONSUMPTION SCHEDULING problem for cases where only $a_{\max}$ is small. When $\mathrm{nr} = 1$ and $a_{\max} = 1$, then we have polynomial-time solvability via Theorem 3.1. The next theorem shows that this extends to the case of constant values of nr and $a_{\max}$ if we assume unary encoding. To obtain this result, we develop a dynamic-programming-based algorithm for $1, \mathrm{NI} \mid \mathrm{nr} = 1 \mid -$ and apply Lemma 3.1.

Theorem 4.1 $1 \mid \mathrm{nr} = \mathrm{const}, \mathrm{unary} \mid C_{\max}$ *can be solved in* $O(q \cdot a_{\max}^{O(1)} \cdot n^{O(a_{\max})} \cdot \log u_{\max})$ *time.*

Proof We will describe a dynamic-programming procedure that computes whether there exists a gapless schedule (Lemma 3.1). Let r be the (constant) number of different resources. We distinguish jobs by their processing times as well as their resource requirements. To this end, we define the *type* of a job J_j as a vector $(a_{1,j}, a_{2,j}, \ldots, a_{r,j})^T$ containing all its resource requirements. Let $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ be the set of all types such that for all $t \in \mathcal{T}$ there is at least one job J_j of type t. Let $s := |\mathcal{T}|$ and note that $s \leq (a_{\max} + 1)^r$. We first sort all jobs by their type using bucket sort in $O(n)$ time and then sort each of the buckets with respect to the processing times in $O(n \log n)$ time using merge sort. For the sake of simplicity, we will use $P_t[k]$ to describe the set of the k longest jobs of type t. We next define the ℓth *phase* as the time interval from the start of any possible schedule up to $u_{\ell+1}$. Next, we present a dynamic program

$$T : [q] \times [n_{t_1}] \cup \{0\} \times \ldots \times [n_{t_s}] \cup \{0\} \to \{\text{true}, \text{false}\},$$

where n_{t_i} is the number of jobs of type t_i. We want to store true in $T[i, x_1, \ldots, x_s]$ if and only if it is possible to schedule at least the jobs in $P_{t_k}[x_k]$ such that all of these jobs start within the ith phase and there is no gap. If $T[q, n_{t_1}, \ldots, n_{t_s}]$ is true, then this corresponds to a gapless schedule of all jobs and hence this is a solution. We will fill up the table T by increasing values of the first argument. That is, we will first compute all entries $T[1, x_1, x_2, \ldots, x_s]$ for all possible combinations of values for $x_i \in [n_{t_i}] \cup \{0\}$. For the first phase, observe that $T[1, x_1, x_2, \ldots, x_s]$ is set to true if and only if the two following conditions are met. First, there are enough resources available at the start to schedule all jobs in all $P_{t_i}[x_i]$. Second, the sum of all processing times of all “selected jobs” without the longest one ends at least one time step before u_2 (such that the job with the longest processing time can then be started at latest in time step $u_2 - 1$ which is the last time step in the first phase). For increasing values of the first argument, we do the following to compute $T[i, x_1, x_2, \ldots, x_s]$. For all tuples of numbers $(y_1, y_2, \ldots, y_s)$ with $y_k \leq x_k$ for all $k \in [s]$, we check (i) whether $T[i-1, y_1, y_2, \ldots, y_s]$ is true, (ii) whether the corresponding schedule can be extended to a schedule for $T[i, x_1, x_2, \ldots, x_s]$, (iii) whether there are enough resources available, and (iv) whether all selected jobs except for the longest one can be finished at least one time step before $u_{i+1} - 1$. Since checks (i), (iii), and (iv) are very simple, we focus on (ii). To this end, it is actually enough to check whether $\sum_{k \in [s]} \sum_{j \in P_{t_k}[y_k]} p_j \geq u_i - 1$ since if this was not the case but there still was a gapless schedule, then we could add some other job to the $(i-1)$st phase and since we iterate over all possible combinations of values of y_i, we would find this schedule in another iteration.

It remains to analyze the running time of this algorithm. First, the number of table entries in T is upper-bounded by $q \cdot \prod_{k \in [s]} (n_{t_k} + 1) \leq (q \cdot (n+1)^s)$. For each table entry $T[i, x_1, x_2, \ldots, x_s]$, there are at most

$$\prod_{k \in [s]} (x_k + 1) \leq \prod_{k \in [s]} (n_{t_k} + 1) \in O((n+1)^s)$$

possible tuples $(y_1, y_2, \ldots, y_s)$, $y_k \leq x_k$ for all $k \in [s]$ and checks (i) to (iv) can be performed in $O(s)$ time. Thus, the overall running time for computing a gapless schedule is

$$O(q \cdot s \cdot (n+1)^{2s}) \subseteq O(q \cdot (a_{\max}+1)^r \cdot (n+1)^{2r(a_{\max}+1)})$$

and the time for solving $1 \mid \mathrm{nr} = \mathrm{const}, \mathrm{unary} \mid C_{\max}$ is by Lemma 3.1

$$O(q \cdot (a_{\max}+1)^r \cdot (n+1)^{2r(a_{\max}+1)} \cdot \log u_{\max}).$$

□

The question whether $1 \mid \mathrm{nr} = \mathrm{const}, \mathrm{unary} \mid C_{\max}$ is in FPT or whether it is W[1]-hard with respect to $a_{\max}$ remains open even for only a single resource.

We continue with showing that already with two resources and unit processing times of the jobs, Material Consumption Scheduling becomes computationally intractable, even when parameterized by the number of supply dates. Note that NP-hardness for $1 \mid \mathrm{nr} = 2, p_j = 1 \mid C_{\max}$ can also be transferred from Grigoriev et al. (2005, Theorem 4) (the statement is for a different optimization goal but the proof still works).

Proposition 4.1 $1 \mid \mathrm{nr} = 2, p_j = 1 \mid C_{\max}$ *is* W[1]*-hard when parameterized by the number of supply dates even if all numbers are encoded in unary.*

Proof Given an instance I of Unary Bin Packing with k bins, each of size B, and n objects $o_1, o_2, \ldots, o_n$ of sizes $s_1, s_2, \ldots, s_n$ such that $\sum_{i=1}^{n} s_i = kB$, we construct an instance I' of $1 \mid \mathrm{nr} = 2, p_j = 1 \mid C_{\max}$ as follows.

For each $i \in [k]$, we add a supply

$$q_i = (u_i, \tilde{b}_{1,i}, \tilde{b}_{2,i}) := ((i-1)B, B, B(B-1));$$

thus, we create k supply dates. For each object o_i, we create an *object job* $j_i = (p_i, a_{1,i}, a_{2,i}) := (1, s_i, B - s_i)$. Additionally, we create $kB - n$ *dummy jobs*; each of them having processing time 1, no requirement of the resources of the first type, and requiring B resources of the second type. We refer to the constructed instance as I'.

We show that I is a yes-instance if and only if there is a schedule for the constructed instance I' with makespan exactly kB. Clearly, if I is a yes-instance, then there is a k-partition $S_1, S_2, \dots, S_k$ of the objects such that for each $i \in k$, $\sum_{o_j \in S_i} s_j = B$. For some set S_i, we schedule all jobs in S_i, one after another, starting at time $(i-1)B$. Then, in the second step, we schedule the remaining dummy jobs such that we obtain a gapless schedule σ. Naturally, since each S_i contains at most B objects, the object jobs are non-overlapping in σ. Also, because in the second step we have exactly $kB - n$ jobs available (recall that n is the number of object jobs), σ is a gapless schedule with makespan exactly kB. To check the feasibility of σ, let us consider the first B time steps. Note that from time 0 to time $|S_1|$, the machine processes all object jobs representing objects from S_1; from time $|S_1|+1$ to B it processes exactly $B-|S_1|$ dummy jobs. Thus, the number of used resources of type 1 is $\sum_{o_j \in S_1} s_j = B$, and the number of used resources of type 2 is $\sum_{o_j \in S_1}(B-s_j)+(B-|S_1|)B = |S_1|B-B+B^2-|S_1|B = B(B-1)$. As a result, at time $B-1$ there are no resources available, so we can apply the argument for the first B time steps to all following $k-1$ periods of B time steps; we eventually obtain that σ is a feasible schedule.

For the reverse direction, let σ be a schedule with makespan kB for instance I'. We again consider the first B time steps. Let $\mathcal{J}_\text{p}$ be the set of exactly B jobs processed in this time. Let A_1 and A_2 be the usage of the resource of, respectively, type 1 and type 2 by the jobs in $\mathcal{J}_\text{p}$. We denote by $\mathcal{J}_\text{o} \subseteq \mathcal{J}_\text{p}$ the object jobs within $\mathcal{J}_\text{p}$. Then, $A_1 := \sum_{J_j \in \mathcal{J}_\text{o}} a_{1,j} + \sum_{J_j \in \mathcal{J}_\text{p} \setminus \mathcal{J}_\text{o}} a_{1,j}$. In fact, since no dummy job has a requirement for the resources of the first type, we have $A_1 = \sum_{J_j \in \mathcal{J}_\text{o}} a_{1,j}$. Moreover, there are only B resources of type 1 available in the first B time steps, so it is clear that $A_1 \le B$. Using the fact that, for each job j_i, it holds that $a_{2,i} = B - a_{1,i}$, we obtain the following:

$$\begin{aligned} A_2 &:= \sum_{J_j \in \mathcal{J}_\text{o}} a_{2,j} + \sum_{J_j \in \mathcal{J}_\text{p} \setminus \mathcal{J}_\text{o}} a_{2,j} \\ &= |\mathcal{J}_\text{o}|B - A_1 + |\mathcal{J}_\text{p} \setminus \mathcal{J}_\text{o}|B = B^2 - A_1. \end{aligned}$$

Using the above relation between A_1 and A_2, we show that $A_1 = B$. For the sake of contradiction, assume that $A_1 < B$. Immediately, we obtain that

$$A_2 > B^2 - B = B(B-1),$$

which is impossible since we only have $B(B-1)$ resources of the second type in the first B time steps of schedule σ. Thus, since $A_1 = B$, we use exactly B resources of the first type and exactly $(B-1)B$ resources of the second type in the first B time steps of σ. We can repeat the whole argument for all following $k-1$ periods of B time steps. Eventually, we obtain a solution to I by taking the objects corresponding to the object jobs scheduled in the subsequent periods of B-time steps.

The reduction is clearly applicable in polynomial time and the number of supply dates is a function depending solely on the number of bins in the input instance of Unary Bin Packing. □

Proposition 4.1 limits the hope for obtaining positive results for the general case with multiple resources. Still, when adding the number of different resources to the combined parameter, we can extend our fixed-parameter tractability result from Theorem 3.3. Since we expect the number of different resources to be rather small in real-world applications, we consider this result to be of practical interest.

Proposition 4.2 $1, \text{NI} \mid \text{nr} = r \mid C_{\max}$ *is fixed-parameter tractable for the parameter* $q + a_{\max} + r$, *where* q *is the number of supplies and* $a_{\max}$ *is the maximum resource requirement of a job.*

Proof The main observation needed to extend the MILP from Theorem 3.3 is that, by Lemma 3.2, given two jobs with the same resource requirement, there is always a schedule that first schedules the longer (dominating) job. In essence, for each phase and each possible resource requirement, a solution is still fully described by the respective number of jobs with that requirement scheduled in the phase.

For multiple resources, we describe the resource requirement of job J_j by a *resource vector*

$$\mathbf{s} = (a_{1,j}, a_{2,j}, \dots, a_{r,j}).$$

We use the following non-negative integer variables:

1. $x_{w,\mathbf{s}}$ denoting the number of jobs with resource vector $\mathbf{s}$ being started in phase w,
2. $x^{\Sigma}_{w,\mathbf{s}}$ denoting the number of jobs with resource vector $\mathbf{s}$ being started between phase 1 and w,
3. $\alpha_{y,w}$ denoting the number of resources of type y available in the beginning of phase w,
4. d_w denoting the endpoint of phase w, that is, the time when the job started latest in phase w ends.

All constraints and proof arguments translate in a straightforward way from the proof of Theorem 3.3. □

Next, by a reduction from Independent Set we show that Material Consumption Scheduling is intractable for an unbounded number of resources even when combining all other considered parameters.

Theorem 4.2 $1|\,\mathrm{nr},\, p_j = 1|C_{\max}$ *is* NP*-hard and* W[1]*-hard parameterized by* $u_{\max}$ *even if* $a_{\max} = \tilde{b}_{\max} = 1$ *and* $q = 2$.

Proof We provide a parameterized reduction from the NP-hard Independent Set problem which, given an undirected graph G and a positive integer k, asks if there is an *independent set* of size k, that is, a set of k vertices in G which are pairwise non-adjacent. Independent Set is W[1]-hard with respect to the size k of the independent set (Downey & Fellows, 2013).

Given an Independent Set instance (G, k), we create an instance of $1|\,\mathrm{nr},\, p_j = 1|C_{\max}$ as follows. Let $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$. For each edge in G, we create one resource, that is, $\mathrm{nr} = m$. At time $u_1 = 0$, we provide one initial unit of every resource. At time $u_2 = k$, we provide another unit of every resource. For each vertex $v_j \in V(G)$, there is one job J_j of length one with a resource requirement being consistent with the incident edges, that is, $a_{i,j} = 1$ if $v_j \in e_i$ and $a_{i,j} = 0$ if $v_j \notin e_i$.

We claim that there is an independent set of size k if and only if there is a schedule with makespan $C_{\max} = n$.

For the "if" direction, let $V' = \{v_{\ell_1}, \ldots, v_{\ell_k}\}$ be an independent set in G. Observe that every schedule that schedules the jobs $J_{\ell_1}, \ldots, J_{\ell_k}$ (in any order) in the first phase and then all other jobs (in any order) in the second phase is feasible and has makespan $C_{\max} = n$. Feasibility comes from the fact that we have an independent set so that no two jobs scheduled in the first phase require the same resource. (After the second supply, there are enough resources to schedule all jobs.) The makespan can be verified by observing that we have unit processing time and that the machine does not idle.

For the "only if" direction, assume that there is a feasible schedule with makespan $C_{\max} = n$ and let $\mathcal{J}' = \{J_{j_1}, \ldots, J_{j_k}\}$ be the jobs which are scheduled at time points 0 to $k - 1$. (Note that $\mathcal{J}'$ is well-defined since each job has length one, there are n jobs in total, and the makespan is n.) We claim that the set $V' = \{v_{j_1}, \ldots, v_{j_k}\}$ is an independent set. Assume toward a contradiction that two vertices are adjacent. Then, both jobs are scheduled in the first phase and require one unit of the the same edge resource; a contradiction. □

Finally, to complement Theorem 4.2, we show that $1|\,\mathrm{nr}\,|C_{\max}$ parameterized by $u_{\max}$ is in XP. Note that for this algorithm we do not need to assume unit processing times.

Proposition 4.3 $1|\,\mathrm{nr}\,|C_{\max}$ *is solvable in* $O(n^{u_{\max}+1} \cdot \log u_{\max})$ *time.*

Proof We solve $1|\,\mathrm{nr}\,|C_{\max}$ by basically brute-forcing all schedules up to time $u_{\max}$. By Lemma 3.1, we may assume that we are looking for a gapless schedule. The algorithm now iteratively guesses the next job in the schedule (starting with a first job at time 0). Since we assume that processing any job requires at least one time unit, the algorithm guesses at most $u_{\max}$ jobs until time $u_{\max}$. Afterwards, we can schedule the jobs in any order as no new resources become available afterwards. Notice that guessing up to $u_{\max}$ jobs takes $O(n^{u_{\max}})$ time, verifying whether a schedule is feasible takes $O(n)$ time, and Lemma 3.1 adds an additional $\log u_{\max}$ factor for assuming a gapless schedule. This results in an overall running time of $O(n^{u_{\max}+1} \cdot \log u_{\max})$. □

5 Conclusion

We provided a seemingly first thorough multivariate complexity analysis of the problem of Material Consumption Scheduling on a single machine. Our main focus was the case of one resource type ($\mathrm{nr} = 1$). Table 1 surveys our results.

Open research questions refer to the parameterized complexity for the single parameters $a_{\max}$ and $p_{\max}$, their combination, and the closely related parameter number of job types. Notably, this might be challenging to answer because these questions are closely related to long-standing open questions for Bin Packing and $P||C_{\max}$ (Knop & Koutecký, 2018; Knop et al., 2020; Mnich & vanBevern, 2018). Indeed, parameter combinations may be unavoidable in order to identify practically relevant tractable cases. For example, it is not hard to derive from our statements (particularly Assumption 1 and Lemma 3.1) fixed-parameter tractability for $\tilde{b}_{\max} + q$ while for the single parameters $\tilde{b}_{\max}$ and q it is both times computationally hard.

Another challenge is to study the case of multiple machines, which is obviously computationally at least as hard as the case of a single machine but relevant in practice. It seems, however, far from obvious to generalize our algorithms to the multiple-machines case.

We have also seen that cases where the jobs can be ordered with respect to the domination ordering (Definition 3.1) are polynomial-time solvable. It may be promising to consider structural parameters measuring the distance from this tractable special case in the spirit of distance from triviality parameterization (Guo et al., 2004; Niedermeier, 2006).

Our results for multiple resources certainly represent only first steps. They clearly invite further investigations, particularly concerning a multivariate complexity analysis. It would also be interesting to study other objectives than the makespan in the future.

Acknowledgements We thank the anonymous reviewers of the *Journal of Scheduling* and *AAAI '21* for their valuable feedback. The main work was done while Robert Bredereck and Andrzej Kaczmarczyk were with TU Berlin. The project started while Robert Bredereck, Péter Györgyi, and Rolf Niedermeier were attending the Lorentz center workshop "Scheduling Meets Fixed-Parameter Tractability", February 4–8, 2019, Leiden, the Netherlands, organized by Nicole Megow, Matthias Mnich, and Gerhard J. Woeginger. Without this meeting, this work would not have been started. Andrzej Kaczmarczyk was mostly supported by the

DFG project AFFA (BR 5207/1 and NI 369/15) and partly by the funds of Polish Ministry of Science and Higher Education assigned to AGH University of Science and Technology. Péter Györgyi was supported by the National Research, Development and Innovation Office grant no. TKP2021- NKTA-01 and by the Janos Bolyai Research Scholarship of the Hungarian Academy of Sciences. We dedicate this paper to Rolf, who tragically passed away recently. We are deeply affected by this loss of our co-author, colleague, and advisor. Rolf contributed tremendously to computer science and, in particular, to multivariate algorithms, and should have continued doing so for a long time. The computer science community shall build on the foundations he has laid.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Belkaid, F., Maliki, F., Boudahri, F., & Sari, Z. (2012). A branch and bound algorithm to minimize makespan on identical parallel machines with consumable resources. In: *Proceedings of the 3rd International Conference on Mechanical and Electronic Engineering*, (pp. 217–221). Springer.

Bentert, M., Bredereck, R., Györgyi, P., Kaczmarczyk, A., & Niedermeier, R. (2021). A multivariate complexity analysis of the material consumption scheduling problem. In: *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, (pp. 11755–11763). AAAI Press.

Bentert, M., van Bevern, R., & Niedermeier, R. (2019). Inductive k-independent graphs and c-colorable subgraphs in scheduling: A review. *Journal of Scheduling, 22*(1), 3–20.

Bérczi, K., Király, T., & Omlor, S. (2020). Scheduling with nonrenewable resources: Minimizing the sum of completion times. In: *Proceedings of the 6th International Symposium on Combinatorial Optimization*, (pp. 167–178). Springer.

Bodlaender, H. L., & Fellows, M. R. (1995). W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters, 18*(2), 93–97.

Bodlaender, H.L., & van der Wegen, M. (2020). Parameterized complexity of scheduling chains of jobs with delays. In: *Proceedings of the 15th International Symposium on Parameterized and Exact Computation (IPEC '20)*, (pp. 4:1–4:15). Schloss Dagstuhl - Leibniz-Zentrum für Informatik

Bredereck, R., Faliszewski, P., Niedermeier, R., Skowron, P., & Talmon, N. (2020). Mixed integer programming with convex/concave constraints: Fixed-parameter tractability and applications to multicovering and voting. *Theoretical Computer Science, 814*, 86–105.

Carlier, J. (1984). Problèmes d'ordonnancements à contraintes de ressources: Algorithmes et complexité. Thèse d'état. Université Paris 6

Carrera, S., Ramdane-Cherif, W., & Portmann, M.C. (2010). Scheduling supply chain node with fixed component arrivals and two partially flexible deliveries. In: *Proceedings of the 5th International Conference on Management and Control of Production and Logistics (MCPL '10)*,(p. 6). IFAC Publisher.

Carlier, J., & Rinnooy Kan, A. H. G. (1982). Scheduling subject to nonrenewable resource constraints. *Operations Research Letters, 1*, 52–55.

Chrétienne, P. (2008). On single-machine scheduling without intermediate delays. *Discrete Applied Mathematics, 156*(13), 2543–2550.

Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., & Saurabh, S. (2015). *Parameterized Algorithms*. Springer.

Davari, M., Ranjbar, M., De Causmaecker, P., & Leus, R. (2020). Minimizing makespan on a single machine with release dates and inventory constraints. *European Journal of Operational Research, 286*(1), 115–128.

Downey, R. G., & Fellows, M. R. (2013). *Fundamentals of Parameterized Complexity*. Springer.

Fellows, M. R., & McCartin, C. (2003). On the parametric complexity of schedules to minimize tardy tasks. *Theoretical Computer Science, 298*(2), 317–324.

Flum, J., & Grohe, M. (2006). *Parameterized Complexity Theory*. Springer.

Frank, A., & Tardos, É. (1987). An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica, 7*(1), 49–65.

Ganian, R., Hamm, T., & Mescoff, G. (2020). The complexity landscape of resource-constrained scheduling. In: *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, (pp. 1741–1747). International Joint Conferences on Artificial Intelligence Organization

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326.

Grigoriev, A., Holthuijsen, M., & van de Klundert, J. (2005). Basic scheduling problems with raw material constraints. *Naval Research of Logistics, 52*, 527–553.

Guo, J., Hüffner, F., & Niedermeier, R. (2004). A structural view on parameterizing problems: Distance from triviality. In: *Proceedings of the First International Workshop on Parameterized and Exact Computation*, (pp. 162–173). Berlin Heidelberg:Springer.

Györgyi, P., & Kis, T. (2014). Approximation schemes for single machine scheduling with non-renewable resource constraints. *Journal of Scheduling, 17*, 135–144.

Györgyi, P., & Kis, T. (2015). Approximability of scheduling problems with resource consuming jobs. *Annals of Operations Research, 235*(1), 319–336.

Györgyi, P., & Kis, T. (2015). Reductions between scheduling problems with non-renewable resources and knapsack problems. *Theoretical Computer Science, 565*, 63–76.

Györgyi, P., & Kis, T. (2017). Approximation schemes for parallel machine scheduling with non-renewable resources. *European Journal of Operational Research, 258*(1), 113–123.

Györgyi, P., & Kis, T. (2019). Minimizing total weighted completion time on a single machine subject to non-renewable resource constraints. *Journal of Scheduling, 22*(6), 623–634.

Györgyi, P., & Kis, T. (2022). New complexity and approximability results for minimizing the total weighted completion time on a single machine subject to non-renewable resource constraints. *Discrete Applied Mathematics, 311*, 97–109.

Heeger, K., Hermelin, D., Mertzios, G.B., Molter, H., Niedermeier, R., & Shabtay, D. (2021). Equitable scheduling on a single machine. In: *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, (pp. 11818–11825). AAAI Press.

Hermelin, D., Kubitza, J. M., Shabtay, D., Talmon, N., & Woeginger, G. J. (2019). Scheduling two agents on a single machine: A parameterized analysis of NP-hard problems. *Omega, 83*, 275–286.

Hermelin, D., Manoussakis, G., Pinedo, M., Shabtay, D., & Yedidsion, L. (2020). Parameterized multi-scenario single-machine scheduling problems. *Algorithmica, 82*(9), 2644–2667.

Hermelin, D., Pinedo, M., Shabtay, D., & Talmon, N. (2019). On the parameterized tractability of single machine scheduling with rejection. *European Journal of Operational Research, 273*(1), 67–73.

Hermelin, D., Shabtay, D., & Talmon, N. (2019). On the parameterized tractability of the just-in-time flow-shop scheduling problem. *Journal of Scheduling, 22*(6), 663–676.

Herr, O., & Goel, A. (2016). Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints. *European Journal of Operational Research, 248*(1), 123–135.

Jansen, K., Kratsch, S., Marx, D., & Schlotter, I. (2013). Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences, 79*(1), 39–49.

Kannan, R. (1987). Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research, 12*(3), 415–440.

Knop, D., & Koutecký, M. (2018). Scheduling meets n-fold integer programming. *Journal of Scheduling, 21*(5), 493–503.

Knop, D., Koutecký, M., & Mnich, M. (2020). Combinatorial n-fold integer programming and applications. *Mathematical Programming, 184*(1), 1–34.

Lenstra, H. W., Jr. (1983). Integer programming with a fixed number of variables. *Mathematics of Operations Research, 8*(4), 538–548.

Mnich, M., & van Bevern, R. (2018). Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research, 100*, 254–261.

Mnich, M., & Wiese, A. (2015). Scheduling and fixed-parameter tractability. *Mathematical Programming, 154*(1–2), 533–562.

Niedermeier, R. (2006). *Invitation to Fixed-Parameter Algorithms*. Oxford University Press.

Slowinski, R. (1984). Preemptive scheduling of independent jobs on parallel machines subject to financial constraints. *European Journal of Operational Research, 15*, 366–373.

Toker, A., Kondakci, S., & Erkip, N. (1991). Scheduling under a non-renewable resource constraint. *Journal of the Operational Research Society, 42*, 811–814.

van Bevern, R., Mnich, M., Niedermeier, R., & Weller, M. (2015). Interval scheduling and colorful independent sets. *Journal of Scheduling, 18*(5), 449–469.

van Bevern, R., Niedermeier, R., & Suchý, O. (2017). A parameterized complexity view on non-preemptively scheduling interval-constrained jobs: Few machines, small looseness, and small slack. *Journal of Scheduling, 20*(3), 255–265.

Xie, J. (1997). Polynomial algorithms for single machine scheduling problems with financial constraints. *Operations Research Letters, 21*(1), 39–42.