Scheduling Meets Fixed-Parameter Tractability

Matthias Mnich\textsuperscript{1} and Andreas Wiese\textsuperscript{2}

\textsuperscript{1} Cluster of Excellence MMCI, Saarbrücken, Germany. m.mnich@mmci.uni-saarland.de
\textsuperscript{2} Max Planck Institute for Computer Science, Saarbrücken, Germany. awiese@mpi-inf.mpg.de

Abstract. Fixed-parameter tractability analysis and scheduling are two core domains of combinatorial optimization which led to deep understanding of many important algorithmic questions. However, even though fixed-parameter algorithms are appealing for many reasons, no such algorithms are known for many fundamental scheduling problems.

In this paper we present the first fixed-parameter algorithms for classical scheduling problems such as makespan minimization, scheduling with job-dependent cost functions—one important example being weighted flow time—and scheduling with rejection. To this end, we identify crucial parameters that determine the problems’ complexity. In particular, we manage to cope with the problem complexity stemming from numeric input values, such as job processing times, which is usually a core bottleneck in the design of fixed-parameter algorithms. We complement our algorithms with \textit{W[1]}-hardness results showing that for smaller sets of parameters the respective problems do not allow \textit{FPT}-algorithms. In particular, our positive and negative results for scheduling with rejection explore a research direction proposed by Dániel Marx [1].

We hope that our contribution yields a new and fresh perspective on scheduling and fixed-parameter algorithms and will lead to further fruitful interdisciplinary research connecting these two areas.

1 Introduction

Scheduling and fixed-parameter tractability are two very well-studied research areas. In scheduling, the usual setting is that one is given a set of machines and a set of jobs with individual characteristics. The jobs need to be scheduled on the machines according to some problem-specific constraints, such as release dates, precedence constraints, or rules regarding preemption and migration. Typical objectives are minimizing the global makespan, the weighted sum of completion times of the jobs, or the total flow time. During the last decades of research on scheduling, many important algorithmic questions have been settled. For instance, for minimizing the makespan and the weighted sum of completion time on identical machines, (1 + \(\epsilon\))-approximation algorithms (PTASs) are known for almost all \textit{NP}-hard settings [23].

However, the running time of these approximation schemes usually has a bad dependence on \(\epsilon\), and in practise exact algorithms are often desired. These considerations motivate to study which scheduling problems are \textit{fixed-parameter tractable} (\textit{FPT}), which amounts to identifying instance-dependent parameters \(k\) that allow for algorithms that find optimal solutions in time \(f(k) \cdot n^{O(1)}\) for instances of size \(n\) and some function \(f\) depending only on \(k\). Separating the dependence of \(k\) and \(n\) is often much more desirable than a running time of, e.g., \(O(n^k)\), which becomes infeasible even for small \(k\) and large \(n\). The parameter \(k\) measures the complexity of a given instance and thus, problem classification according to parameters yields an instance-depending measure of problem hardness.

Despite the fundamental nature of scheduling problems, and the clear advantages of fixed-parameter algorithms, to the best of our knowledge no such algorithms are known for the classical scheduling problems we study here. One obstacle towards obtaining positive results appears to be that—in contrast to most problems known to be fixed-parameter tractable—scheduling problems involve many numerical input data (e.g., job processing times, release dates, job weights), which alone render many problems \textit{NP}-hard, thus ruling out fixed-parameter algorithms. One contribution of this paper is that—for the fundamental problems studied here—choosing the number of distinct numeric values or an upper bound on them as the parameter suffices to overcome this impediment. Note that this requirement is much weaker than assuming the parameter to be bounded by a constant (that can appear in the exponent of the running time), as e.g., a running time
same length are almost equally distributed among the machines, up to an additive error term of first prove that for any number of machines, we can restrict ourselves to (optimal) solutions where jobs of the parameter defines an upper bound on the job processing times appearing in an instance with n jobs. We first prove that for any number of machines, we can restrict ourselves to (optimal) solutions where jobs of the same length are almost equally distributed among the machines, up to an additive error term of ± f(k) jobs. This insight can be used as an independent preprocessing routine which optimally assigns the majority of the jobs of an instance (given that n ≫ k). After this preparation, we show that the remaining problem can be formulated as an integer program in fixed dimension, yielding an overall running time of f(k) ⋅ n^O(1). We note that without a fixed parameter, the problem is strongly NP-hard. For the much more general machine model of unrelated machines, we show that R||C_{\text{max}} is fixed-parameter tractable when choosing the number of machines and the number of distinct processing times as parameters. We reduce this problem again to integer programming in fixed dimension where our variables model how many jobs of each size are scheduled on each machine. To ensure that an assignment of all given jobs to these “slots” exists we argue via Hall’s Theorem and ensure that for each subset of jobs there are enough usable slots. While there the number of such subsets is exponentially large, we identify a compact set of constraints that alone ensure that for the computed slots on the machines a feasible job assignment exists. Let us remark that our problem is general enough that we do not see a way of using the “number of numbers” result by Fellows et al. [4]. Note that if the number of machines or the number of processing times are constant, the problem is still NP-hard [5], and thus no FPT-algorithms can exist for those cases.

Then, in Section 3 we study scheduling with rejection. Each job j is specified by a processing time p_j, a weight w_j, and a rejection cost e_j (all jobs are released at time zero). We want to reject a set J’ of at most k jobs, and schedule all other jobs on one machine to minimize \( \sum_{j \in J'} w_j C_j + \sum_{j \in J'} e_j \). For a given instance, we identify three key parameters: the number of distinct processing times, the number of distinct weights, and the maximum number k of jobs to be rejected. We show that if any two of the three values are taken as parameters, the problem becomes fixed-parameter tractable. If k and either of the other two are parameters, then we show that an optimal solution is characterized by one of sufficiently few possible patterns of jobs to be rejected. Once we guessed the correct pattern, an actual solution can be found by a dynamic program efficiently. If the number of distinct processing times and lengths are parameters (but not k), we provide a careful modeling of the problem as an integer program with convex objective function in fixed dimension. Here, we need to take particular care to incorporate the rejection costs as bounded-degree polynomials, to be able to use the deep theory of solving convex programs in fixed dimension efficiently. To the best of our knowledge, this is the first time that convex programming is used in fixed-parameter algorithms. We complement this result by showing that if only the number of rejected jobs k is the fixed parameter, then the problem becomes W[1]-hard, which prohibits the existence of a fixed-parameter algorithm, unless FPT = W[1] (which would imply subexponential time algorithms for many canonical NP-complete problems). In particular, this justifies choosing not only k but also the number of distinct processing times or weights as a parameter. Our results respond to a quest by Marx [1] for investigating the fixed-parameter tractability of scheduling with rejection.

Finally, in Section 4 we turn our attention to the parametrized dual of the latter problem: scheduling with rejection of at least n − s jobs (s being the parameter). We reduce this to a much more general problem which can be cast as the profit maximization version of the General Scheduling Problem (GSP) [6]. We need to select a subset J’ of at most s jobs to schedule from a given set J, and each scheduled job j yields a profit \( f_j(C_j) \), depending on its completion time \( C_j \). Note that this function can be different for each job. Additionally, each
job $j$ has a release date $r_j$ and a processing time $p_j$. The goal is to schedule these jobs on one machine to maximize $\sum_{j \in J'} f_j(C_j)$. We study the preemptive as well as the non-preemptive version of this problem. In its full generality, GSP is not well understood. Despite that, we are able to give a fixed-parameter algorithm if the number of distinct processing times is bounded by a parameter, as well as the maximum cardinality of $J'$. This justifies to consider additionally the number of distinct processing times as a parameter. On the other hand, if only the latter is the parameter the problem is para-NP-hard. In particular, this includes functions that stem from difficult scheduling objectives such as weighed flow time, weighted tardiness, and weighted flow time squared. We complement our findings by showing that if only the number of scheduled jobs $s$ is a fixed parameter, the problem is W[1]-hard. This justifies to consider additionally the number of distinct processing times as a parameter. On the other hand, if only the latter is the parameter the problem is para-NP-hard, as it is NP-hard if $p_j \in \{1, 3\}$ for each job $j$. In particular, we prove that for this setting the ordinary GSP is NP-hard, which might be of independent interest.

| Problem | Parameters | Result |
|---------|------------|--------|
| $P||C_{max}$ | maximum $p_j$ | FPT |
| $R||C_{max}$ | #distinct $p_j$ and #machines | FPT |
| $1|r_j,(pmtn)|\max\sum_{s \leq j} f_s(C_j)$ | #selected jobs $s$ and #distinct $p_j$ | FPT |
| $1|r_j,(pmtn)|\max\sum_{s \leq j} f_s(C_j)$ | #selected jobs $s$ | W[1]-hard |
| $1|r_j,(pmtn)|\max\sum_{s \leq j} f_s(C_j)$ | #distinct $p_j$ (in fact $\forall p_j \in \{1, 3\}$) | para-NP-hard |
| $1||\sum_{s \leq k} e_j + \sum w_j C_j$ | #rejected jobs $k$ and #distinct $p_j$ | FPT |
| $1||\sum_{s \leq k} e_j + \sum w_j C_j$ | #rejected jobs $k$ and #distinct $w_j$ | FPT |
| $1||\sum_{s \leq k} e_j + \sum w_j C_j$ | #distinct $p_j$ and #distinct $w_j$ | FPT |
| $1||\sum_{s \leq k} e_j + \sum w_j C_j$ | #rejected jobs $k$ | W[1]-hard |

Table 1. Summary of our results.

Our contributions are summarized in Table 1.

### 1.2 Related Work

**Scheduling.** One very classical scheduling problem studied in this paper is to schedule a set of jobs non-preemptively on a set of $m$ identical machines, i.e., $P||C_{max}$. Research for it dates back to the 1960s when Graham showed that the greedy list scheduling algorithm yields a $(2 - \frac{1}{m})$-approximation and a 4/3-approximation when the jobs are ordered non-decreasingly by length [7]. After a series of improvements [8][9][10][11], Hochbaum and Shmoys present a polynomial time approximation scheme (PTAS), even if the number of machines is part of the input [3]. On unrelated machines, the problem is NP-hard to approximate with a better factor than 3/2 [12][13] and there is a 2-approximation algorithm [3] that extends to the generalized assignment problem [13]. For the restricted assignment case, i.e., each job has a fixed processing time and a set of machines where one can assign it to, Svensson [14] gives a polynomial time algorithm that estimates the optimal makespan up to a factor of 33/17 + $\epsilon \approx 1.9412 + \epsilon$.

For scheduling jobs with release dates preemptively on one machine, a vast class of important objective functions is captured by the General Scheduling Problem (GSP). In its full generality, it is studied by Bansal and Pruhs [6] who give a $O(\log \log nP)$-approximation, improving on several earlier results for special cases. One particularly important special case is the weighted flow time objective where previously to Bansal and Pruhs [6] the best known approximation factors where $O(\log^2 P)$, $O(\log W)$, and $O(\log nP)$ [15][16]; here, $P$ and $W$ are the maximum ratios of job processing times and weights, respectively. Also, a quasi-PTAS with running time $n^{O(\log P \cdot \log W)}$ is known [17].
A generalization of classical scheduling problems is scheduling with rejection. There, each job $j$ is additionally equipped with a rejection cost $e_j$. The scheduler has the freedom to reject job $j$ and to pay a penalty of $e_j$, in addition to some (ordinary) objective function for the scheduled jobs. For one machine and the objective being to minimize the sum of weighted completion times, Engels et al. [18] give an optimal pseudopolynomial dynamic program for the case that all jobs are released at time zero and show that the problem is weakly NP-hard. Sviridenko and Wiese [19] give a PTAS for arbitrary release dates. For the scheduling objective being the makespan and given multiple machines, Hoogeveen et al. [20] give FPTASs for almost all machine settings, and a 1.58-approximation for the APX-hard case of an arbitrary number of unrelated machines (all results when allowing preemption).

**Fixed-Parameter Tractability.** Until now, to the best of our knowledge, no fixed-parameter algorithms for the classical scheduling problems studied in this paper have been devised. Though restricted scheduling problems have been considered by the parameterized complexity community, this generally meant that jobs are represented as vertices of a graph with conflicting jobs connected by an edge, and then one finds a maximum independent set in the graph; examples are given by van Bevern et al. [21]. In contrast, classical scheduling problems investigated in the framework of parameterized complexity appear to be intractable; for example, $k$-processor scheduling with precedence constraints is W[2]-hard and scheduling unit-length tasks with deadlines and precedence constraints and $k$ tardy tasks is W[1]-hard [23], for parameter $k$.

Mere exemptions seem to be an algorithm by Marx and Schlotter [24] for makespan minimization where $k$ jobs have processing time $p \in \mathbb{N}$ and all other jobs have processing time 1, for combined parameter $(k, p)$, and work by Chu et al. [25] who consider checking feasibility of a schedule (rather than optimization).

A potential reason for this lack of positive results (fixed-parameter algorithms) might be that the knowledge of fixed-parameter algorithms for weighted problems is still in a nascent stage, and scheduling problems are inherently weighted, having job processing times, job weights, etc. We remark though that some scheduling-type problems can be addressed by choosing as parameter the “number of numbers”, as done by Fellows et al. [4].

## 2 Minimizing the Makespan

In this section we consider the problem $P||C_{\text{max}}$, i.e., where a given a set $J$ of $n$ jobs (all released at time zero) must be scheduled non-preemptively on a set of $m$ identical machines, as to minimize the makespan of the schedule. We develop a fixed-parameter algorithm that solves this problem in time $f(p_{\text{max}}) \cdot n^{O(1)}$, so our parameter is the maximum processing time $p_{\text{max}}$ over all jobs.

In the sequel, we say that some job $j$ is of type $t$ if $p_j = t$; let $J_t := \{ j \in J \mid p_j = t \}$. First, we prove that there is always an optimal solution in which each machine has almost the same number of jobs of each type, up to an additive error of $\pm f(p_{\text{max}})$ for suitable function $f$. This allows us to fix some jobs on the machines. For the remaining jobs, we show that each machine receives at most $2f(p_{\text{max}})$ jobs of each type; hence there are only $(2f(p_{\text{max}}))^p_{\text{max}}$ possible configurations for each machine. We solve the remaining problem with an integer linear program in fixed dimension.

As a first step, for each type $t$, we assign \( \left\lfloor \frac{|J_t|}{m} \right\rfloor - f(p_{\text{max}}) \) jobs of type $t$ to each machine; let $J_0 \subseteq J$ be this set of jobs. This is justified by the next lemma, which can be proven by starting with an arbitrary optimal schedule and exchanging jobs carefully between machines until the claimed property holds.

**Lemma 1.** There is a function $f : \mathbb{N} \to \mathbb{N}$ with $f(p_{\text{max}}) = 2^{O(p_{\text{max}} \cdot \log p_{\text{max}})}$ for all $p_{\text{max}} \in \mathbb{N}$ such that every instance of $P||C_{\text{max}}$ admits an optimal solution in which for each type $t$, each of the $m$ machines schedules at least $\left\lfloor \frac{|J_t|}{m} \right\rfloor - f(p_{\text{max}})$ and at most $\left\lfloor \frac{|J_t|}{m} \right\rfloor + f(p_{\text{max}})$ jobs of type $t$.

**Proof.** For each machine $i$ denote by $J_t^{(i)}$ the set of jobs of type $t$ scheduled on $i$ in some (optimal) schedule. We prove the following (more technical) claim: there always exists an optimal solution in which for every pair of machines $i$ and $i'$, and for each $\ell \in \{1, \ldots, p_{\text{max}}\}$, we have that $||J_t^{(i)}| - |J_t^{(i')}|| \leq h(\ell) \cdot g(p_{\text{max}})$, where $g(p_{\text{max}}) := (p_{\text{max}})^3 + p_{\text{max}}$ and $h(\ell) = \sum_{j=\ell}^{p_{\text{max}}} j \cdot h(j)$.
Suppose that there are two machines $i, i'$ such that there is a value $\ell$ for which $|J^{(i)}_\ell| - |J^{(i')}_{\ell'}| > h(\ell) \cdot g(p_{\max})$. Assume, without loss of generality, that $|J^{(i)}_\ell| - |J^{(i')}_{\ell'}| > h(\ell) \cdot g(p_{\max})$ and $|J^{(i)}_{\ell'}| - |J^{(i')}_{\ell''}| \leq h(\ell') \cdot g(p_{\max})$ for all $\ell' > \ell$ and all machines $i, i'$. We show that $\ell'$ has at least a certain volume of jobs which are shorter than $\ell$. We calculate that

$$
\sum_{j=1}^{\ell-1} j \cdot |J^{(i)}_j| = \left( \sum_{j=1}^{p_{\max}} j \cdot |J^{(i)}_j| \right) - \ell \cdot |J^{(i')}_{\ell'}| - \left( \sum_{j=\ell+1}^{p_{\max}} j \cdot |J^{(i')}_{\ell'}| \right)
$$

$$
\geq \left(-p_{\max} + \sum_{j=1}^{p_{\max}} j \cdot |J^{(i)}_j| \right) - \ell \cdot |J^{(i')}_{\ell'}| - \left( \sum_{j=\ell+1}^{p_{\max}} j \cdot |J^{(i')}_{\ell'}| \right)
$$

$$
\geq \left(-p_{\max} + \sum_{j=1}^{\ell} j \cdot |J^{(i)}_j| \right) - \ell \cdot |J^{(i')}_{\ell'}| - \left( \sum_{j=\ell+1}^{p_{\max}} j \cdot |J^{(i')}_{\ell'}| + h(j) \cdot g(p_{\max}) \right)
$$

$$
> -p_{\max} + \sum_{j=1}^{\ell} j \cdot |J^{(i)}_j| + \ell \cdot (h(\ell) \cdot g(p_{\max}) - |J^{(i)}_{\ell'}|) - \sum_{j=\ell+1}^{p_{\max}} j \cdot h(j) \cdot g(p_{\max})
$$

$$
= -p_{\max} + \sum_{j=1}^{\ell-1} j \cdot |J^{(i)}_j| + \ell \cdot h(\ell) \cdot g(p_{\max}) - \sum_{j=\ell+1}^{p_{\max}} j \cdot h(j) \cdot g(p_{\max})
$$

$$
\geq (p_{\max})^3,
$$

where the first inequality stems from the fact that the considered schedule is optimal and therefore the loads of any two machines can differ by at most $p_{\max}$. Hence, there must be at least one type $\ell' < \ell$ such that $\ell' \cdot |J^{(i')}_{\ell'}| \geq (p_{\max})^2$, and thus $|J^{(i')}_{\ell'}| \geq p_{\max}$. Since the least common multiple of any two values in $\{1, \ldots, p_{\max}\}$ is at most $(p_{\max})^2$, we can swap $r \in \{1, \ldots, p_{\max}\}$ jobs of type $\ell$ from machine $M_i$ with $r' \in \{1, \ldots, p_{\max}\}$ jobs of type $\ell'$ from machine $M_{i'}$, without changing the total load of any of the two machines. By continuing inductively we obtain a schedule satisfying $|J^{(i)}_{\ell'}| - |J^{(i')}_{\ell''}| \leq h(\ell') \cdot g(p_{\max})$ for any two machines $i, i'$ and any type $\ell' \geq \ell$. Inductively, we obtain an optimal schedule satisfying the claim for all types $\ell$. Thus, the claim of the lemma follows for an appropriately chosen function $f$.

We now investigate in more detail which choice of $f$ is suitable. To this end, consider first the function $h$ as recursively defined above. Notice that $h$ is parametrized by $\ell$, but also depends on $p_{\max}$. We prove, by induction on $\ell$, that $h(\ell) = \frac{(p_{\max}+1)!}{(\ell+1)!}$. The base case is for $\ell = p_{\max}$, and it follows from the definition that $h(p_{\max}) = 1 = \frac{(p_{\max}+1)!}{(p_{\max}+1)!}$. For the inductive step, let $\ell < p_{\max}$ and take as inductive hypothesis that $h(\ell') = \frac{(p_{\max}+1)!}{(\ell'+1)!}$ for all $\ell' \in \{\ell+1, \ldots, p_{\max}\}$. We then have

$$
h(\ell) = 1 + \sum_{j=\ell+1}^{p_{\max}} j \cdot h(j) = 1 + (\ell+1)h(\ell+1) + \sum_{j=\ell+2}^{p_{\max}} j \cdot h(j)
$$

$$
= 1 + (\ell+1)h(\ell+1) + (h(\ell+1) - 1) = (\ell+2)h(\ell+1)
$$

$$
= (\ell+2)\frac{(p_{\max}+1)!}{(\ell+2)!} = (\ell+2)\frac{(p_{\max}+1)!}{(\ell+1)!(\ell+2)} = \frac{(p_{\max}+1)!}{(\ell+1)!},
$$

as claimed.

Thus, the claim of the lemma holds for $f(p_{\max}) := h(1) \cdot g(p_{\max})$, and therefore we have $f(p_{\max}) \leq (p_{\max}+1)!((p_{\max})^3 + p_{\max}) = 2^{O(p_{\max} \log p_{\max})}$.

Denote by $J' = J \setminus J_0$ be the set of yet unscheduled jobs. We ignore all other jobs from now on. By Lemma [1] there is an optimal solution in which each machine receives at most $2 \cdot f(p_{\max}) + 1$ jobs from each type. Hence, there are at most $(2 \cdot f(p_{\max}) + 2)^{p_{\max}}$ ways how the schedule for each machine can look like (up
to permuting jobs of the same length). Therefore, the remaining problem can be solved with the following integer program. Define a set \( \mathcal{C} = \{0, \ldots, 2 \cdot f(p_{\text{max}}) + 1\}^{p_{\text{max}}} \) of at most \((2 \cdot f(p_{\text{max}}) + 2)^{p_{\text{max}}} \) “configurations”, where each configuration is a vector \( C \in \mathcal{C} \) encoding the number of jobs from \( J' \) of each type assigned to a machine.

In any optimal solution for \( J' \), the makespan is in the range \( \{\lceil p(J')/m \rceil, \ldots, \lceil p(J')/m \rceil + p_{\text{max}} \} \), as \( p_j \leq p_{\text{max}} \) for each \( j \). For each value \( T \) in this range we try whether \( \text{opt} \leq T \), where \( \text{opt} \) denotes the minimum makespan of the instance. So fix a value \( T \). We allow only configurations \( C = (c_1, \ldots, c_{p_{\text{max}}}) \) which satisfy \( \sum_{t=1}^{p_{\text{max}}} c_t \cdot i \leq T \); let \( \mathcal{C}(T) \) be the set of these configurations. We define an integer program. For each \( C \in \mathcal{C}(T) \), introduce a variable \( y_C \) for the number of machines with configuration \( C \) in the solution. (As the machines are identical, only the number of machines following each configuration is important.)

\[
\sum_{C \in \mathcal{C}(T)} y_C \leq m \tag{1}
\]

\[
\sum_{C = (c_1, \ldots, c_{p_{\text{max}}}) \in \mathcal{C}(T)} y_C \cdot c_p \geq |J' \cap J|_p, \quad p = 0, \ldots, p_{\text{max}} \tag{2}
\]

\[
y_C \in \{0, \ldots, m\}, \quad C \in \mathcal{C}(T) \tag{3}
\]

Inequality (1) ensures that at most \( m \) machines are used, inequalities (2) ensure that all jobs from each job type are scheduled. The above integer program (1)–(3) has at most \((2 \cdot f(p_{\text{max}}) + 2)^{p_{\text{max}}} \) dimensions.

To determine feasibility of (1)–(3), we employ deep results about integer programming in fixed dimension. As we will need it later, we cite here an algorithm that can even minimize over convex spaces described by (quasi-)convex polynomials, rather than only over polytopes. Known efficient algorithms for convex integer minimization in fixed dimension use variants of Lenstra’s algorithm \( [20] \) for integer programming. The first algorithm of this kind was given by Khachiyan \( [27] \), later improved by Heinz \( [28] \). We use the algorithm by Heinz as stated in a survey by Köppe \( [29] \) Theorem 6.4.

**Theorem 1.** \( [28,29] \). Let \( f, g_1, \ldots, g_m \in \mathbb{Z}[x_1, \ldots, x_t] \) be quasi-convex polynomials of degree at most \( d \geq 2 \), whose coefficients have a binary encoding length of at most \( \ell \). There is an algorithm that in time \( m \cdot \ell^O(1) \cdot d^{O(t)} \cdot 2^{O(t^2)} \) computes a minimizer \( x^* \in \mathbb{Z}^t \) of binary encoding size \( \ell \cdot d^{O(t)} \) for following problem (4), or reports that no minimizer exists.

\[
\min f(x_1, \ldots, x_t) \quad \text{subject to} \quad g_i(x_1, \ldots, x_t) \leq 0, \quad i = 1, \ldots, m \quad x \in \mathbb{Z}^t. \tag{4}
\]

Thus, we can determine feasibility of (1)–(3) in time \( g(p_{\text{max}}) \cdot (\log n + \log m)^{O(1)} \), for suitable function \( g \). The smallest value \( T \) for which it is feasible gives the optimal makespan and together with the preprocessing routine due to Lemma \( [1] \) this yields an optimal schedule.

**Theorem 2.** There is a function \( f \) such that instances of \( P||C_{\text{max}} \) with \( n \) jobs and \( m \) machines can be solved in time \( f(p_{\text{max}}) \cdot (n + \log m)^{O(1)} \).

Recall that without choosing a parameter, problem \( P||C_{\text{max}} \) is strongly \( \text{NP-hard} \) (as it contains \( 3\)-\textsc{Partition}). A natural extension to consider is the problem \( P||C_{\text{max}} \) parameterized by the number \( p \) of distinct processing times. Unfortunately, for this extension Lemma \( [1] \) is no longer true. Let \( q_1, q_2 \) be two different (large) prime numbers. Consider an instance with two identical machines \( M_1, M_2 \), and \( q_1 \) many jobs with processing time \( q_2 \), and similarly \( q_2 \) many jobs with processing time \( q_1 \). The optimal makespan is \( T := q_1 \cdot q_2 \) which is achieved by assigning all jobs with processing time \( q_1 \) on \( M_1 \) and all other jobs on \( M_2 \), giving both machines a makespan of exactly \( T \). However, apart from swapping the machines this is the only optimal schedule since the equation \( q_1 \cdot q_2 = x_1 \cdot q_1 + x_2 \cdot q_2 \) only allows integral solutions \((x_1, x_2)\) such that \( x_1 \) is a multiple of \( q_2 \) and \( x_2 \) is a multiple of \( q_1 \).

Indeed, for constantly many processing times the problem was only recently shown to be polynomial time solvable for any constant \( p \) \( [30] \).
2.1 Bounded Number of Unrelated Machines

We study the problem $Rm||C_{\text{max}}$ where now the machines are unrelated, meaning that a job can have different processing times on different machines. In particular, it might be that a job cannot be processed on some machine at all, i.e., has infinite processing time on that machine. We choose as parameters the number $p$ of distinct (finite) processing times and the number of machines $m$ of the instance.

We model this problem as an integer program in fixed dimension. Denote by $q_1, \ldots, q_p$ the distinct finite processing times in a given instance. For each combination of a machine $i$ and a finite processing time $q_\ell$ we introduce a variable $y_{i,\ell} \in \{0, \ldots, n\}$ that models how many jobs of processing time $q_\ell$ are assigned to $i$. Note that the number of these variables is bounded by $m \cdot p$. An assignment to these variables can be understood as allocating $y_{i,\ell}$ slots for jobs with processing time $q_\ell$ to machine $i$, without specifying what actual jobs are assigned to these slots. Assigning the jobs to the slots can be understood as a bipartite matching: introduce one vertex $v_j$ for each job $j$, one vertex $w_{s,i}$ for each slot $s$ on each machine $i$, and an edge $\{v_j, w_{s,i}\}$ whenever job $j$ has the same size on machine $i$ as slot $s$. According to Hall’s Theorem, there is a matching in which each job is matched if and only if for each set of jobs $J' \subseteq J$ there are at least $|J'|$ slots to which at least one job in $J'$ can be assigned. For one single set $J'$ the latter can be expressed by a linear constraint; however, the number of subsets $J'$ is exponential. We overcome this as follows: We say that two jobs $j, j'$ are of the same type if $p_{i,j} = p_{i,j'}$ for each machine $i$. Note that there are only $(\bar{p} + 1)^m$ different types of jobs. As we will show, it suffices to add a constraint for sets of jobs $J'$ such that for each job type either all or none of the jobs of that type are contained in $J'$ (those sets “dominate” all other sets). This gives rise to the following integer program. Denote by $Z$ the set of all job types and for each $z \in Z$ denote by $J_z \subseteq J$ the set of jobs of type $z$. For each set $Z' \subseteq Z$ denote by $Q_{i,z'}$ the set of distinct finite processing times of jobs of types in $Z'$ on machine $i$.

\[
\begin{align*}
\min \ T & \quad \text{s.t.} \quad \sum_{\ell \in \{1, \ldots, p\}} y_{i,\ell} \cdot q_\ell \leq T, \quad i = 1, \ldots, m \tag{5} \\
\sum_{z \in Z'} |J_z| & \leq \sum_{\ell \in \{1, \ldots, p\}} \sum_{i \in Q_{i,z'}} y_{i,\ell} \quad \forall \ Z' \subseteq Z \tag{6} \\
y_{i,\ell} & \in \{0, \ldots, n\} \quad i = 1, \ldots, m, \ \ell = 1, \ldots, p \tag{7} \\
T & \geq 0 \tag{8}
\end{align*}
\]

As the above IP has only $m \cdot \bar{p}$ dimensions and $2(\bar{p} + 1)^m + m$ constraints, we solve it using Theorem 1 Finding then a bipartite matching for the actual assignment of the jobs can be done in polynomial time. This yields the following theorem.

**Theorem 3.** Instances of $Rm||C_{\text{max}}$ with $m$ machines and $n$ jobs with $p$ distinct finite processing times can be solved in time $f(\bar{p}, m) \cdot (n + \log \max_\ell q_\ell)^{O(1)}$ for a suitable function $f$.

**Proof.** We show that the integer program (5)-(8) has a solution of value $T$ if and only if there is a solution with makespan $T$. First suppose that there is a solution with makespan $T$. Then for each machine $i$ and each processing time $q_\ell$ we set $y_{i,\ell}$ to be the number of jobs with processing time $q_\ell$ on machine $i$. As the makespan is $T$, we have that $\sum_{\ell \in \{1, \ldots, p\}} y_{i,\ell} \cdot q_\ell \leq T$ for each machine $i$. Now consider a constraint for a set $Z' \subseteq Z$. All jobs in $\cup_{z \in Z'} J_z$ are assigned in the given solution. Therefore, we must have that $\sum_{z \in Z'} |J_z| \leq \sum_{i} \sum_{\ell \in Q_{i,z}} y_{i,\ell}$ since the right-hand side of the inequality denotes the number of slots (given by the $y_{i,\ell}$ variables) that at least one job in $\cup_{z \in Z'} J_z$ can be assigned to.

Conversely, suppose that the IP (5)-(8) has a solution of value $T$. We prove that in polynomial time we can construct a feasible schedule with makespan $T$. We construct a bipartite graph as described above: we introduce a vertex $v_j$ for each job $j$. For each machine $i$ and each processing time $q_\ell$, we introduce $y_{i,\ell}$ slots of size $q_\ell$ and add a vertex $w_{s,i}$ for each slot $s$ on machine $i$. For each pair of a job $j$ and a slot $s$ on a machine $i$, we introduce an edge $\{v_j, w_{s,i}\}$ if and only if $p_{i,j}$ equals the size of slot $s$. According to Hall’s Theorem, there is a matching that matches each vertex $v_j$ if and only if for each set of jobs $J' \subseteq J$ there are at least $|J'|$ slots to which at least one job in $J'$ can be assigned. Consider a set $J' \subseteq J$ and let $\text{slot}(J')$ be the set of slots
that a job in $J'$ can be assigned to. Denote by $\text{full}(J')$ the set of all jobs $j \in J$ for which there is a job $j' \in J'$ such that $j$ and $j'$ are of the same type. Observe that $J' \subseteq \text{full}(J')$ and $\text{slot}(J') = \text{slot}(\text{full}(J'))$. Then there is a set $Z \subseteq Z$ such that $\bigcup_{z \in Z} J_z = \text{full}(J')$. Due to the constraint $\sum_{z \in Z'} |J_z| \leq \sum_i \sum_{j \in Q_i} y_{i,j,t}$ there are enough slots for the jobs in $\bigcup_{z \in Z'} J_z = \text{full}(J')$. As $J' \subseteq \text{full}(J')$ and $\text{slot}(J') = \text{slot}(\text{full}(J'))$ there are also enough slots for the jobs in $J'$. Applying this reasoning to all sets $J' \subseteq J$, due to Hall’s Theorem there exists a schedule with makespan at most $T$. Using standard bipartite matching theory (see e.g. [31, Chapter 16]) we can find such a matching in polynomial time.

A natural question is the case when only the number of machines is a fixed parameter. If the job processing times can be arbitrary, then already $P2||C_{\text{max}}$ is NP-hard due to the contained PARTITION problem. If one additionally requires the job processing times to be polynomially bounded, then $P||C_{\text{max}}$ is still W[1]-hard if only the number of machines is a parameter, due to a result by Jansen et al. [32].

On the other hand, $R||C_{\text{max}}$ is NP-hard if only processing times $\{1, 2, \infty\}$ are allowed [12]. This justifies to take both $m$ and $\bar{p}$ as a parameters in the unrelated machine case.

3 Scheduling with Rejection

In this section we study scheduling with rejection to optimize the weighted sum of completion time plus the total rejection cost, i.e. $1|\sum_{i \in J} e_{ij} + \sum w_j C_j$. Formally, we are given an integer $k$ and a set $J$ of $n$ jobs, all released at time zero. Each job $j \in J$ is characterized by a processing time $p_j \in \mathbb{N}$, a weight $w_j \in \mathbb{N}$ and rejection cost $e_j \in \mathbb{N}$. The goal is to reject a set $J' \subseteq J$ of at most $k$ jobs and to schedule all other jobs non-preemptively on a single machine, as to minimize $\sum_{j \in J \setminus J'} w_j C_j + \sum_{j \in J'} e_j$. Note that if rejection is not allowed ($k = 0$), the problem is solved optimally by scheduling jobs according to non-decreasing Smith ratios $w_j/p_j$, breaking ties arbitrarily [33].

3.1 Parameters Number of Rejected Jobs and Distinct Processing Times or Weights

Denote by $\bar{p} \in \mathbb{N}$ the number of distinct processing times in a given instance. First, we assume that $\bar{p}$ and the maximum number $k$ of rejected jobs are parameters. Thereafter, using a standard reduction, we will derive an algorithm for the case that $k$ and the number $\bar{W}$ of distinct weights are parameters.

Denote by $q_1, \ldots, q_{\bar{p}}$ the distinct processing times in a given instance. For each $i \in \{1, \ldots, \bar{p}\}$, we guess the number of jobs with processing time $q_i$ which are rejected in an optimal solution. Each possible guess is characterized by a vector $v = \{v_1, \ldots, v_{\bar{p}}\}$ whose entries $v_i$ contain integers between $0$ and $k$, and whose total sum is at most $k$. There are at most $(k + 1)^{\bar{p}}$ such vectors $v$, each one prescribing that at most $v_i$ jobs of processing time $p_i$ can be rejected. We enumerate them all. One of these vectors must correspond to the optimal solution, so the reader may assume that we know this vector $v$.

In the following, we will search for the optimal schedule that respects $v$, meaning that for each $i \in \{1, \ldots, \bar{p}\}$ at most $v_i$ jobs of processing time $q_i$ are rejected. To find an optimal schedule respecting $v$, we use a dynamic program. Suppose the jobs in $J$ are labeled by $1, \ldots, n$ by non-increasing Smith ratios $w_j/p_j$. Each dynamic programming cell is characterized by a value $n' \in \{0, \ldots, n\}$, and a vector $v'$ with $\bar{p}$ entries which is dominated by $v$, meaning that $v'_i \leq v_i$ for each $i \in \{1, \ldots, \bar{p}\}$. For each pair $(n', v')$ we have a cell $C(n', v')$ modeling the following subproblem. Assume that for jobs in $J' := \{1, \ldots, n'\}$ we have already decided whether we want to schedule them or not. For each processing time $q_i$ denote by $n'_i$ the number of jobs in $J'$ with processing time $q_i$. Assume that for each type $i$, we have decided to reject $v_i - n'_i$ jobs from $J'$. Note that then the total processing time of the scheduled jobs sums up to $t := \sum_i q_i \cdot (n'_i - (v_i - n'_i))$. It remains to define a solution for the jobs in $J'' := \{n' + 1, \ldots, n\}$ during time interval $[t, \infty)$, such that for each type $i$ we can reject up to $v'_i$ jobs.

Clearly, the cell $C(0, v)$ contains the optimal value to the entire instance. Also, by definition, $C(n, v') = 0$ for any $v'$, since then $J'' = \emptyset$. For the dynamic programming transition, we prove:

\[\text{\vdots}\]
Lemma 2. Let \( C(n', v') \) be a cell and let \( \text{opt}(n', v') \) be the optimal solution value to its subproblem. Let \( i \in \{1, \ldots, p\} \) be such that \( p_{n' + 1} = q_i \), and let \( t := \sum_i q_i \cdot (v_i - (v_i')) \). If \( v_i' = 0 \) then
\[
\text{opt}(n', (v_1', \ldots, v_i', \ldots, v_p')) = \text{opt}(n' + 1, (v_1', \ldots, v_i', \ldots, v_p')) + (t + p_{n' + 1}) \cdot w_{n'},
\]
otherwise
\[
\text{opt}(n', (v_1', \ldots, v_i', \ldots, v_p')) = \\
\min \left\{ \text{opt}(n' + 1, (v_1', \ldots, v_i', \ldots, v_p')) + (t + p_{n' + 1}) \cdot w_{n'}, \text{opt}(n' + 1, (v_1', \ldots, v_i' - 1, \ldots, v_p')) + e_{n' + 1} \right\}.
\]

Proof. First, suppose that \( \text{opt}(n', v') \) schedules job \( n' \). Due to our sorting, \( n' \) has the largest Smith ratio of all jobs in \( \{n', \ldots, n\} \), and an optimal schedule with value \( \text{opt}(n', v') \) schedules \( n' \) at time \( t \) at cost \( (t + p_{n'}) \cdot w_{n'} \), and the optimal solution value to the remaining subproblem is given by \( \text{opt}(n' + 1, (v_1', \ldots, v_i', \ldots, v_p')) \). If a solution of value \( \text{opt}(n', v') \) rejects \( n' \) (which can only happen if \( v_i' > 0 \)), then \( \text{opt}(n', v') = \text{opt}(n' + 1, (v_1', \ldots, v_i' - 1, \ldots, v_p')) + e_{n'} \).

On the other hand, there is a solution for cell \( C(n', v') \) given by scheduling \( n' \) at time \( t \) and all other non-rejected jobs in the interval \( \lfloor t + p_{n'} \rfloor \) at cost \( \text{opt}(n' + 1, (v_1', \ldots, v_i', \ldots, v_p')) + (t + p_{n'}) \cdot w_{n'} \). If \( v_i' > 0 \) there is also a feasible solution with cost \( \text{opt}(n' + 1, (v_1', \ldots, v_i - 1, \ldots, v_p')) + e_{n'} \) which rejects job \( n' \) and schedules all other non-rejected jobs during the interval \( [t, \infty) \).

The size of the dynamic programming table is bounded by \( n \cdot (k + 1)^p \). Hence, the whole dynamic program runs in fixed-parameter time, proving that:

Theorem 4. For sets \( J \) of \( n \) jobs with \( p \) distinct processing time, the problem \( 1 \| \sum_j e_j + \sum w_j C_j \) is solvable in time \( O(n \cdot (k + 1)^p + n \cdot \log n) \).

For the problem \( 1 \| \sum w_j C_j \), it is known [34] Theorem 3.1] that one can interchange weights and processing times, bijectively mapping feasible solutions preserving their costs. Hence, we obtain for parameters \( k \) and \( \pi \) distinct job weights:

Corollary 1. For sets \( J \) of \( n \) jobs with \( \pi \) distinct job weights, the problem \( 1 \| \sum_j e_j + \sum w_j C_j \) is solvable in time \( O(n \cdot (k + 1)^\pi + n \cdot \log n) \).

We show that when only the number \( k \) of rejected jobs is taken as parameter, problem becomes \( W[1] \)-hard. This justifies to define additionally the number of weights or processing times as parameter. We remark that when jobs have non-trivial release dates, then even for \( k = 0 \) the problem is \( \text{NP-hard} \) [35].

Theorem 5. Problem \( 1 \| \sum_j e_j + \sum w_j C_j \) is \( W[1] \)-hard if the parameter is the number \( k \) of rejected jobs.

Proof. We reduce from \( k\text{-SUBSET SUM} \), for which the input consists of integers \( s_1, \ldots, s_n \in \mathbb{N} \) and two values \( k, q \in \mathbb{N} \). The goal is to select a subset of \( k \) of the given integers \( s_i \) that sum up to \( q \). Parameterized by \( k \), this problem is known to be \( W[1] \)-hard [36]. Our reduction mimics closely a reduction from (ordinary) \( \text{SUBSET SUM} \) in [13] that shows that \( 1 \| \sum_j e_j + \sum w_j C_j \) is weakly \( \text{NP-hard} \).

Suppose we are given an instance of \( k\text{-SUBSET SUM} \). Let \( S = \sum_{i=1}^n s_i \). We construct an instance of \( 1 \| \sum_j e_j + \sum w_j C_j \) with \( n \) jobs, where each job \( j \) has processing time \( p_j := s_j \), weight \( w_j := s_j \), rejection cost \( e_j := (S - q) \cdot s_j + \frac{1}{2} s_j^2 \). Since \( p_j = w_j \) for all jobs \( j \), the ordering of the scheduled jobs \( J \setminus J' \) does not affect the value of \( \sum_{j \in J \setminus J'} w_j C_j \). Using this fact and substituting for the rejection penalty, we can rewrite
the objective function as follows:

\[
\sum_{j \in J \setminus J'} w_j C_j + \sum_{j \in J'} e_j = \sum_{j \in J \setminus J'} s_j + \sum_{i, j \in J \setminus J'} s_i + \sum_{j \in J'} e_j
\]

\[
= \sum_{j \in J \setminus J'} s_j^2 + \sum_{j < i, j \in J \setminus J'} s_j s_i + \sum_{j \in J'} (S - q) \cdot s_j + \frac{1}{2} s_j^2
\]

\[
= \frac{1}{2} \left( \sum_{j \in J \setminus J'} s_j \right)^2 + \sum_{j \in J \setminus J'} s_j^2 + (S - q) \sum_{j \in J'} s_j + \frac{1}{2} \sum_{j \in J'} s_j^2.
\]

Since \( \sum_{j=1}^n s_j^2 \) does not depend on the choice of \( J' \), this is equivalent to minimizing the following function \( h(x) \), with \( x = \sum_{j \in J \setminus J'} s_j \):

\[
h(x) := \frac{1}{2} x^2 + (S - q) (S - x) = \frac{1}{2} \left( \sum_{j \in J \setminus J'} s_j \right)^2 + (S - q) \left( S - \sum_{j \in J \setminus J'} s_j \right).
\]

The unique minimum of \( h(x) \) is \( \frac{1}{2} S^2 - \frac{1}{2} q^2 \) at \( x = S - q \), i.e., when \( \sum_{j \in J \setminus J'} s_j = S - q \). Hence, if such a set \( J' \) exists, the resulting value is optimal for the scheduling problem. Therefore, if the solution to the created instance of 1\| || \sum \leq k e_j + \sum w_j C_j \) has value less than or equal to \( \frac{1}{2} S^2 + \frac{1}{2} q^2 + \frac{1}{2} \sum_{l=1}^n s_l^2 \), then the instance of \(|\text{SUBSET SUM}| \) is a "yes"-instance. Conversely, if the instance of \(|\text{SUBSET SUM}| \) is "yes", then there is a schedule of value \( \frac{1}{2} S^2 + \frac{1}{2} q^2 + \frac{1}{2} \sum_{l=1}^n s_l^2 \).

### 3.2 Parameter Number of Distinct Processing Times and Weights

We consider the number of distinct processing times and weights as parameters. To this end, we say that two jobs \( j, j' \) are of the same type if \( p_j = p_{j'} \) and \( w_j = w_{j'} \); let \( \tau \) be the number of types in an instance. Note, however, that jobs with the same type might have different rejection costs, so we cannot bound the "number of numbers" in the input by a function of the parameters only—this prohibits that we use a strategy similar to Fellows et al. [4] who solve simpler problems parameterized by the number of distinct input numbers. Instead, we resort to convex integer programming, which is used here for the first time in fixed-parameter algorithms. The running time of our algorithm will depend only polynomially on \( k \), the upper bound on the number of jobs we are allowed to reject. For each type \( i \), let \( w^{(i)} \) be the weight and \( p^{(i)} \) be the processing time of jobs of type \( i \). Assume that job types are numbered \( 1, \ldots, \tau \) such \( \frac{w^{(i)}}{p^{(i)}} \geq \frac{w^{(i+1)}}{p^{(i+1)}} \) for each \( i \in \{1, \ldots, \tau - 1\} \). Clearly, an optimal solution schedules jobs ordered non-increasingly by Smith’s ratio without preemption.

The basis for our algorithm is a problem formulation as a convex integer minimization problem with dimension at most \( 2 \tau \). In an instance, for each \( i \), we let \( n_i \) be the number of jobs of type \( i \) and introduce an integer variable \( x_i \in \mathbb{N}_0 \) modeling how many jobs of type \( i \) we decide to schedule. We introduce the linear constraint \( \sum_{i=1}^\tau (n_i - x_i) \leq k \), to ensure that at most \( k \) jobs are rejected.

The objective function is more involved. For each type \( i \), scheduling the jobs of type \( i \) costs

\[
\sum_{\ell=1} x_i w^{(i)} \cdot (\ell \cdot p^{(i)} + \sum_{\ell' < \ell} x_{i'} \cdot p^{(i')}) = w^{(i)} \cdot x_i \sum_{\ell' < \ell} x_{i'} \cdot p^{(i')} + w^{(i)} \cdot p^{(i)} \sum_{\ell=1} x_i \ell
\]

\[
= w^{(i)} \cdot x_i \sum_{\ell' < \ell} x_{i'} \cdot p^{(i')} + w^{(i)} \cdot p^{(i)} \cdot \frac{x_i (x_i + 1)}{2} =: s_i(x).
\]
Note that $s_i(x)$ is a convex polynomial of degree 2 (being the sum of quadratic polynomials with only positive coefficients). Observe that when scheduling $x_i$ jobs of type $i$, it is optimal to reject the $n_i - x_i$ jobs with lowest rejection costs among all jobs of type $i$. Assume the jobs of each type $i$ are labeled $s_1^{(i)}, \ldots, s_{n_i}^{(i)}$ by non-decreasing rejection costs. For each $s \in \mathbb{N}$ let $f_i(s) := \sum_{t=1}^{n_i-s} e_{j_t}$. In particular, to schedule $x_i$ jobs of type $i$ we can select them such that we need to pay $f_i(s)$ for rejecting the non-scheduled jobs (and this is an optimal decision). The difficulty is that the function $f_i(s)$ is in general not expressible by a polynomial whose degree is globally bounded (i.e., for each possible instance), which prevents a direct application of Theorem 1.

However, in Lemma 3 we show that $f_i(s)$ is the maximum of $n_i$ linear polynomials, allowing us to formulate a convex program and solve it by Theorem 1.

**Lemma 3.** For each type $i$ there is a set of $n_i$ polynomials $p_i^{(1)}, \ldots, p_i^{(n_i)}$ of degree one such that $f_i(s) = \max_{\ell} p_i^{(\ell)}(s)$ for each $s \in \{0, \ldots, n_i\}$.

**Proof.** For each $\ell \in \{0, \ldots, n_i - 1\}$ we define $p_i^{(\ell)}(s)$ to be the unique polynomial of degree one such that $p_i^{(\ell)}(0) = f_i(0)$ and $p_i^{(\ell)}(\ell + 1) = f_i(\ell + 1)$. We observe that $f_i(s)$ is convex, since $f_i(\ell) - f_i(\ell + 1) = e_{j_{n_i-\ell}} \geq e_{j_{n_i-\ell-1}} = f_i(\ell+1) - f_i(\ell+2)$ for each $\ell \in \{0, \ldots, n_i - 2\}$. Therefore, the definition of the polynomials implies that $f_i(s) \geq p_i^{(\ell)}(s)$ for each $\ell \in \{0, \ldots, n_i - 1\}$ and each $s \in \{0, \ldots, n_i\}$. Since for each $s \in \{0, \ldots, n_i - 1\}$ we have that $p_i^{(s)}(s) = f_i(s)$ and $p_i^{(n_i-1)}(n_i) = f_i(n_i)$, we conclude that $f_i(s) = \max_{\ell} p_i^{(\ell)}(s)$ for each $s \in \{0, \ldots, n_i\}$.

Lemma 3 allows modeling the entire problem with the following convex program, where for each type $i$, variable $g_i$ models the rejection costs for jobs of type $i$.

$$
\min \sum_{i=1}^{\tau} g_i + s_i(x) \text{ s.t. } \sum_{i=1}^{\tau} (a_i - x_i) \leq k, g_i \geq p_i^{(\ell)}(x_i) \forall i \in \{1, \ldots, \tau\}, \forall \ell \in \{0, \ldots, n_i\}, g, x \in \mathbb{Z}_{\geq 0}^\tau.
$$

Observe that this convex program admits an optimal solution with $g_i = \max_{\ell} p_i^{(\ell)}(x_i) = f_i(x_i)$ for each $i$. Thus, solving the convex program yields an optimal solution to the overall instance.

**Theorem 6.** For sets of $n$ jobs of $\tau$ types the problem 1||$\sum_{j=1}^{s} e_j + \sum_{j=1}^{n} w_j C_j$ can be solved in time $(n + \log(\max_j \max\{e_j, p_j, w_j\}))^{O(1)} \cdot 2^{O(\tau^3)}$.

**Proof.** The number of integer variables is $2\tau$. All constraints and the objective function can be expressed by convex polynomials of degree at most 2. Hence, using Theorem 1 we obtain a running time of $(1 + n \cdot \tau) \cdot \log(\max_j \max\{e_j, p_j, w_j\})^{O(1)} 2^{O(\tau^3)}$.

## 4 Profit Maximization for General Scheduling Problems

In this section, we consider the parameterized dual problem to scheduling jobs with rejection: We consider the problem to reject at least $n - s$ jobs (s being the parameter) to minimize the total cost given by the rejection penalties plus the cost of the schedule. As we will show, this is equivalent to selecting at most $s$ jobs and to schedule them in order to maximize the profit of the scheduled jobs. In contrast to the previous section, here we allow jobs to have release dates and that the profit of a job depends on the time when it finishes in the schedule, described by a function that might be different for each job (similarly as for job dependent cost functions).

Formally, we are given a set $J$ of $n$ jobs, where each job $j$ is characterized by a release date $r_j$, a processing time $p_j$, and a non-increasing profit function $f_j(t)$. We assume that the functions $f$ are given as oracles and that we can evaluate them in time $O(1)$ per query. Let $\mathcal{P}$ denote the number of distinct processing times $p_j$ in
Lemma 4. Let to maximize \( \sum_{j \in J} f_j(C_j) \), where \( C_j \) denotes the completion time of \( j \) in the computed schedule. We call this problem the \( s \)-bounded General Profit Scheduling Problem, or \( s \)-GPSP for short. Observe that this generic problem definition allows to model profit functions that stem from scheduling objectives such as weighted flow time and weighted tardiness.

We like to note that in a first version of this paper we presented an algorithm using three, rather than two parameters. An anonymous referee gave us advice how to remove one parameter and we present the improved version here. First, we show how to find the best solution with a non-preemptive schedule, and subsequently we show how to extend the algorithm to the preemptive setting.

4.1 Non-Preemptive Schedules

Denote by \( q_1, \ldots, q_s \) the set of arising processing times in the instance. We start with a color-coding step, where we color each job uniformly at random with one color from \( \{1, \ldots, s\} \). When coloring the jobs randomly, with probability \( s!/s^s \) all scheduled in an optimal solution (the set \( J \)) will receive distinct colors. We call such a coloring of \( J \) a good coloring. When trying enough random colorings, with high probability we will find at least one good coloring. We will discuss later how to derandomize this algorithm. For now, assume a fixed good coloring \( c : J \rightarrow \{1, \ldots, s\} \).

We describe now a dynamic program that finds the best non-preemptive schedule of at most \( s \) jobs with different colors. The dynamic program has one cell \( C(S,t) \) for each combination of a set \( S \subseteq \{1, \ldots, s\} \) and a time \( t \in T \), where \( T := \{r_j + \sum_{i=1}^{p} s_i \cdot q_i \mid j \in J, s_1, \ldots, s_p \in \{0, \ldots, s\}\} \). Observe that the set \( T \) contains all possible completion times of a job in an optimal non-preemptive schedule (we will show later that this is even true for preemptive schedules). Also, \( |T| \leq n \cdot (s + 1)^s \). The cell \( C(S,t) \) models the subproblem of scheduling a set of at most \( |S| \) jobs such that their colors are pairwise different and contained in \( S \), and so that the last job finishes by time \( t \) the latest. We will denote by \( \text{opt}(S,t) \) the value of the optimal solution for this subproblem with respect to the fixed coloring \( c \).

Consider an optimal solution with value \( \text{opt}(S,t) \). If no job finishes at time \( t \), then \( \text{opt}(S,t) = \text{opt}(S,t') \) for some \( t' < t \). Else, there is some job \( j \) with \( c(j) \in S \) that finishes at time \( t \); then \( \text{opt}(S,t) = f_j(t) + \text{opt}(S \setminus \{c(j)\}, t') \), where \( t' \) is the largest value in \( T \) with \( t' \leq t - p_j \). We now prove this formally.

**Lemma 4.** Let \( c : J \rightarrow \{1, \ldots, s\} \) be a coloring, let \( S \subseteq \{1, \ldots, s\} \) and let \( t \in T \). The optimal value \( \text{opt}(S,t) \) for the dynamic programming cell \( C(S,t) \) equals

\[
\max \left\{ \max_{t' \in T: t' < t} \text{opt}(S,t'), \quad \max_{j \in J: (r_j \leq t - p_j) \land (c(j) \in S)} f_j(t) + \text{opt}(S \setminus \{c(j)\}, \max\{t' \in T \mid t' \leq t - p_j\}), 0 \right\}.
\]

**Proof.** For cells \( C(S,t_{\min}) \) with \( t_{\min} = \min_{t \in T} t \) the claim is true since \( t_{\min} = \min_j r_j \) and so \( \text{opt}(S,t_{\min}) = 0 \). Now consider a cell \( C(S,t) \) with \( t > t_{\min} \). Let \( \text{opt}'(S,t) \) denote the right-hand side of (9). We show that \( \text{opt}(S,t) \geq \text{opt}'(S,t) \) and \( \text{opt}(S,t) \leq \text{opt}'(S,t) \).

For the first claim, we show that there is a schedule with value \( \text{opt}'(S,t) \). If \( \text{opt}'(S,t) = 0 \) or \( \text{opt}'(S,t) = \max_{t' \in T: t' < t} \text{opt}(S,t') \) then the claim is immediate. Now suppose that there is a job \( j \) with \( r_j \leq t - p_j \) and \( c(j) \in S \) such that \( \text{opt}(S,t) = f_j(t) + \text{opt}(S \setminus \{c(j)\}, \max\{t' \in T \mid t' \leq t - p_j\}) \). Then there is a feasible schedule for the problem encoded in cell \( C(S,t) \) that is given by the jobs in the optimal solution for the cell \( C(S \setminus \{c(j)\}, \max\{t' \in T \mid t' \leq t - p_j\}) \), and additionally schedules job \( j \) during \( [t - p_j, t] \). The profit of this solution is \( f_j(t) + \text{opt}(S \setminus \{c(j)\}, \max\{t' \in T \mid t' \leq t - p_j\}) \).

For showing that \( \text{opt}(S,t) \leq \text{opt}'(S,t) \) consider the schedule for \( \text{opt}(S,t) \) and assume w.l.o.g. that the set \( T \) contains all start and end times of jobs in that schedule. If no job finishes at time \( t \) then \( \text{opt}(S,t) = \max_{t' \in T: t' < t} \text{opt}(S,t') \), using that \( T \) contains all finishing times of a job in the optimal schedule. Note that \( \{t' \in T \mid t' < t\} \neq \emptyset \) since we assumed that \( t > t_{\min} \). If a job \( j \) finishes at time \( t \) then \( \text{opt}(S,t) = f_j(t) + \text{opt}(S \setminus \{c(j)\}, \max\{t' \in T \mid t' \leq t - p_j\}) \). This implies that \( \text{opt}(S,t) \leq \text{opt}'(S,t) \). \( \square \)

To compute the value \( \text{opt}(S,t) \), our dynamic program evaluates expression (9). Note that this transition implies \( \text{opt}(S,t_{\min}) = 0 \) for \( t_{\min} \) the minimum value in \( T \). The number of dynamic programming cells is at
most \( 2^s \cdot |T| = 2^s n \cdot (s+1)^p \). Evaluating expression \( 4 \) takes time \( O(|T| + n) = O((s+1)^p + n) \), given that optimal solutions to all subproblems are known. Together with Lemma \( 4 \) this yields the following lemma.

**Lemma 5.** There is an algorithm that, given an instance of \( s \)-GPSP with a set \( J \) of \( n \) jobs of at most \( p \) distinct processing times together with a good coloring \( c : J \to \{1, \ldots, s\} \), in time \( O(2^s \cdot n^2 \cdot (s+1)^{3p}) \) computes an optimal non-preemptive schedule.

Instead of coloring the jobs randomly, we can use a family of hash functions, as described by Alon et al. \[ 37 \]. Using our notation, they show that there is a family \( F \) of \( 2^{O(s)} \log n \) hash functions \( J \to \{1, \ldots, s\} \) such that for any set \( J' \subseteq J \) (in particular for the set of jobs \( J' \) that is scheduled in an optimal solution) there is a hash function \( F \in \mathcal{F} \) such that \( F \) is bijective on \( J' \). The value of each of these functions on each specific element of \( J \) can be evaluated in \( O(1) \) time. This yields the following.

**Theorem 7.** There is a deterministic algorithm that, given an instance of \( s \)-GPSP with \( n \) jobs and \( p \) processing times, in time \( 2^{O(s)} \cdot (s+1)^{3p} \cdot n^2 \log n \) computes an optimal non-preemptive schedule.

### 4.2 Preemptive Schedules

We now extend our algorithm from the previous section to the setting when preemption of jobs is allowed. Now the dynamic program becomes more complicated. As before, we color all jobs in \( J \) uniformly at random with colors from \( \{1, \ldots, s\} \). Fix a good coloring \( c : J \to \{1, \ldots, s\} \). In the dynamic programming table we have one cell \( C(S,t_1,t_2,P) \) for each combination of a set \( S \subseteq \{1, \ldots, s\} \), values \( t_1, t_2 \in T \), where as above \( T = \{ r_j + \sum_{i=1}^p s_i \cdot q_i \mid j \in J, s_1, \ldots, s_p \in \{0, \ldots, s\} \} \), and a value \( P \in \mathcal{P} := \{ \sum_{i=1}^p s_i \cdot q_i \mid s_1, \ldots, s_p \in \{0, \ldots, s\} \} \) (note that like for \( T \) we have \( |\mathcal{P}| \leq (s+1)^p \)). This cell encodes the problem of selecting at most \( |S| \) jobs with total processing time at most \( P \) such that (i) their colors are pairwise different and contained in \( S \), (ii) they are released during \([t_1, t_2]\); we want to schedule them during \([t_1, t_2]\), possibly with preemption, the objective being to maximize the total profit.

To this end, we first prove that the set \( T \) contains all possible start- and completion times of jobs of an optimal solution. The former denotes the first time in a schedule where each job is being processed.

**Lemma 6.** There is an optimal schedule such that for each scheduled job \( j \), its start time and completion time are both contained in \( T \).

**Proof.** Consider an optimal schedule \( S \) for the set of at most \( k \) jobs that we want to schedule. If we fix the completion times of the jobs in the schedule as deadlines and run the policy of Earliest-Deadline-First (EDF) with these deadlines, then we obtain a schedule \( S' \) where each job finishes no later than in \( S \). This is true since given one machine and jobs with release dates and deadlines, there is a feasible preemptive schedule (i.e., a schedule where each job finishes by its deadline) if and only if EDF finds such a schedule. In particular, since the profit-functions of the jobs are non-increasing, the profit given by \( S' \) is at most the profit given by \( S \).

We prove the claim by induction over the deadlines of the jobs in \( S' \). In fact, we prove the following slightly stronger claim: for the job with \( s' \)-th deadline, its start time and completion time are in the set \( T_{s'} = \{ r_j + \sum_{i=1}^p s_i \cdot q_i \mid j \in J, s_1, \ldots, s_p \in \{0, \ldots, s'\} \} \).

For the job \( j_1 \) with smallest deadline, we have that \( j_1 \) starts at time \( t_{j_1} \in T_1 \) and finishes at time \( t_{j_1} + p_{j_1} \in T_1 \), since \( j_1 \) has highest priority. Suppose by induction that the claim is true for the jobs with the \( s' - 1 \) smallest deadlines, and note that the deadlines are distinct. Consider the job \( j_{s'} \) with the \( s' \)-smallest deadline. Let \( R_{s'} \) denote its start time. Let \( J_{s'} \) denote the jobs finishing before \( R_{s'} \). Similarly, let \( J_{s'} \) denote the jobs finishing during \( (R_{s'} C_{s'}) \). Note that \( j_{s'} \in J_2 \) but \( J_2 \) does not contain a job with deadline later than \( j_{s'} \). Since we run EDF, \( R_{s'} = r_{j_{s'}} \) or \( R_{s'} = C_{j_{s'}} \) for some job \( j_{s'} \). In both cases \( R_{s'} \in \cup_{|J_{s'}|} \), and thus \( R_{s'} = r_{j} + \sum_{i=1}^p s_i^{(1)} \cdot q_i \) for some job \( j \in J \) and values \( s_1^{(1)}, \ldots, s_p^{(1)} \in \{0, \ldots, |J_1|\} \). During \( (R_{s'} C_{s'}) \), at most \(|J_2|\) jobs finish, and their total length is \( \sum_{i=1}^p s_i^{(2)} \cdot q_i \) for values \( s_1^{(2)}, \ldots, s_p^{(2)} \in \{0, \ldots, |J_2|\} \). Hence,

\[
C_{s'} = r_{j} + \sum_{i=1}^p s_i^{(1)} \cdot q_i + \sum_{i=1}^p s_i^{(2)} \cdot q_i = r_{j} + \sum_{i=1}^p s_i \cdot q_i
\]
for some job $j \in J$ and values $s_1, \ldots, s_T \in \{0, \ldots, |J_1| + |J_2|\}$. By $|J_1| + |J_2| = s'$, we have that $C_{j_\rho} \in T_{s'}$. □

Suppose we want to compute an optimal solution of value $\text{opt}(S, t_1, t_2, P)$ for a cell $C(S, t_1, t_2, P)$. In any optimal solution, we can assume that the last finishing job has smallest priority, i.e., we work on this job only if no other job is pending. Let $j$ be this (last) job and let $R_j$ denote its start time. Then, during $[R_j, t_2)$ the only scheduled jobs are $j$ and some jobs that are released during $[R_j, t_2)$. Also, all jobs scheduled during $[t_1, R_j)$ are also released during $[t_1, R_j)$ and they are finished within $[t_1, R_j)$.

This observation gives rise to the dynamic programming transition. Given a cell $C(S, t_1, t_2, P)$, we enumerate all possibilities for the last job $j$, its start time $R_j$, the total processing time of jobs scheduled during $[t_1, R_j)$ and $[R_j, t_2)$, and their colors. One combination corresponds to an optimal solution, and we reduce the overall problem to the respective two subproblems for $[t_1, R_j)$ and $[R_j, t_2)$. (Observe that the color coding step prevents to select the same job in both subproblems.) We justify this transition in the following lemma. For ease of notation, for a cell $C(S, t_1, t_2, P)$ denote by $\text{en}(S, t_1, t_2, P)$ the set of tuples $(j, S', R_j, P', P'')$ that we need to enumerate for splitting the cell, i.e., all such tuples with $c(j) \in S, t_1 \leq r_j \leq t_2 - p_j, S' \subseteq S \setminus \{c(j)\}, t_2 - R_j \geq P'' + p_j, P', P'' \in P$, and $P' + P'' + p_j \leq P$.

**Lemma 7.** For a coloring $c : J \rightarrow \{1, \ldots, s\}$, each dynamic programming cell $C(S, t_1, t_2, P)$ satisfies

$$\text{opt}(S, t_1, t_2, P) = \max \left\{ \max_{t' \in T, t_1 \leq t' < t_2} \text{opt}(S, t_1, t', P), \right.$$  
$$\max_{(j, S', R_j, P', P'') \in \text{en}(S, t_1, t_2, P)} f_j(t_2) + \text{opt}(S', t_1, R_j, P') + \text{opt}(S \setminus (S' \cup \{c(j)\}), R_j, t_2, P''), 0 \right\}.$$  

(10)

**Proof.** Similarly as in the proof of Lemma 3 let $\text{opt}'(S, t_1, t_2, P)$ denote the right-hand side of (10) for the cell $C(S, t_1, t_2, P)$.

Consider a cell $C(S, t_1, t_2, P)$. First, we show that $\text{opt}(S, t_1, t_2, P) \geq \text{opt}'(S, t_1, t_2, P)$. To this end, it suffices to show that there is a feasible solution with value at least $\text{opt}'(S, t_1, t_2, P)$. If $\text{opt}'(S, t_1, t_2, P) = 0$ then the claim is immediate (just take the empty solution). Otherwise, first assume that $\text{opt}'(S, t_1, t_2, P) = \text{opt}(S, t_1, t', P)$ for some $t' < t_2$. Any solution for the cell $C(S, t_1, t', P)$ is also feasible for $C(S, t_1, t_2, P)$, and thus $\text{opt}(S, t_1, t_2, P) \geq \text{opt}(S, t_1, t', P)$. Hence, $\text{opt}(S, t_1, t_2, P) \geq \text{opt}'(S, t_1, t_2, P)$.

Finally, assume that there is a tuple $(j, S', R_j, P', P'')$ such that

$$\text{opt}'(S, t_1, t_2, P) = f_j(t_2) + \text{opt}(S', t_1, R_j, P') + \text{opt}(S \setminus (S' \cup \{c(j)\}), R_j, t_2, P'').$$

We build a solution with this value as follows. During $[t_1, R_j)$, we schedule an optimal solution for the cell $C(S', t_1, R_j, P')$. During $[R_j, t_2)$, we schedule an optimal solution for the cell $C(S \setminus (S' \cup \{c(j)\}), R_j, t_2, P'')$. By assumption, $t_2 - R_j \geq P'' + p_j$. Hence, the schedule for the latter cell leaves total idle time during $[R_j, t_2)$ of at least $p_j$. During these idle times we schedule job $j$ and, thus, $j$ finishes at time $t_2$ the latest. As $S' \subseteq S \setminus \{c(j)\}$ and $S' \cap (S \setminus (S' \cup \{c(j)\})) = \emptyset$, no job is scheduled in both subproblems. Also, $P' + P'' + p_j \leq P$. Hence, the solution is feasible for the cell $C(S, t_1, t_2, P)$ and its value is at least $f_j(t_2) + \text{opt}(S', t_1, R_j, P') + \text{opt}(S \setminus (S' \cup \{c(j)\}), R_j, t_2, P'')$. Thus, $\text{opt}(S, t_1, t_2, P) \geq \text{opt}'(S, t_1, t_2, P)$.

Conversely, we want to show that $\text{opt}(S, t_1, t_2, P) \leq \text{opt}'(S, t_1, t_2, P)$. Consider an optimal solution for the cell $C(S, t_1, t_2, P)$. If $\text{opt}(S, t_1, t_2, P) = 0$, then the claim is immediate. Otherwise, we can assume that in the schedule for the solution, the jobs are ordered according to EDF by their respective completion time (see also the proof of Lemma 6). Let $j$ be the job with largest deadline (and hence smallest priority); then $c(j) \in S$. If $j$ ends at a time $t'^* < t_2$ then $\text{opt}(S, t_1, t'^*, P) = \text{opt}'(S, t_1, t'^*, P) \leq \text{opt}(S, t_1, t_2, P)$.

Now suppose that $j$ finishes at time $t_2$ and let $R_j$ denote its start time in an optimal schedule of value $\text{opt}(S, t_1, t_2, P)$. Observe that $t_1 \leq r_j \leq t_2 - p_j$. Let $S' \subseteq S \setminus \{c(j)\}$ denote the set of colors of jobs scheduled during $[t_1, R_j)$ in an optimal solution with value $\text{opt}(S, t_1, t_2, P)$ (observe that $c(j) \notin S'$), and let $P'$ denote the total processing time of jobs scheduled during $[t_1, R_j)$. Since all jobs scheduled during $[t_1, R_j)$ start and finish during that interval, we have that $P' \in P$. Also, let $P''$ denote the total processing time of jobs other than $j$ that are scheduled during $[R_j, t_2)$ in a solution of value $\text{opt}(S, t_1, t_2, P)$. In particular,
\[ t_2 - R_j \geq P'' + p_j. \] As all those jobs start and finish during \([R_j, t_2]\), this implies that \(P'' \in \mathcal{P}\). Hence, \((j, S', R_j, P', P'') \in \text{en}(S, t_1, t_2, P)\), and thus

\[
\text{opt}(S, t_1, t_2, P) = f_j(t_2) + \text{opt}(S', t_1, R_j, P') + \text{opt}(S \setminus (S' \cup \{c(j)\}), R_j, t_2, P'') \leq \text{opt}'(S, t_1, t_2, P).
\]

Our dynamic program evaluates equation \((\ref{10})\) for each cell. Observe that cells \(C(S, t_1, t_2, P)\) with \(S = \emptyset\) or \(t_1 = t_2\) obey \(\text{en}(S, t_1, t_2, P) = \emptyset\), and so \(\text{opt}(S, t_1, t_2, P) = 0\). This yields the following.

**Lemma 8.** There is an algorithm that, given a set of \(n\) jobs with \(\mathcal{P}\) distinct processing times and a good coloring \(c: J \rightarrow \{1, \ldots, s\}\), in time \(2^{O(n)} n^4\) computes an optimal preemptive schedule for \(s\)-\text{GSP}.

Using the same derandomization technique as above, we obtain the following theorem.

**Theorem 8.** There is a deterministic algorithm that, given a set of \(n\) jobs with \(\mathcal{P}\) distinct processing times, in time \(2^{O(n)} n^4 \log n\) computes an optimal preemptive schedule for \(s\)-\text{GSP}.

Observe that \((\log n)^O(\mathcal{P}) = O((\log \mathcal{P})^O(\mathcal{P}) + n)\); this yields the following.

**Corollary 2.** There is a deterministic algorithm that, given a set of \(n\) jobs with \(\mathcal{P}\) distinct processing times, computes an optimal preemptive schedule for \(O(\log n)\) jobs in time \((\log \mathcal{P})^O(\mathcal{P}) \cdot n^{O(1)}\).

Applying Theorem \(\ref{3}\) we obtain the following corollary when at least \(n - s\) jobs have to be rejected.

**Corollary 3.** There is a deterministic algorithm that, given a set of \(n\) jobs with at most \(\mathcal{P}\) distinct processing times, each job having a private cost function \(w_j\), solves the problem \(1|r_j, (pmtn)|\sum S w_j(C_j) + \sum S e_j\) in time \(2^{O(n)} n^4 \log n\), when at least \(n - s\) jobs have to be rejected.

**Proof.** Any instance of the problem to reject \(n - s\) jobs and schedule the remaining \(s\) jobs to minimize \(1|r_j, (pmtn)|\sum S w_j(C_j) + \sum S e_j\) is equivalent to an instance of \(s\)-\text{GPSP} (with or without preemption, respectively) where for each job \(j\) we define a profit function \(f_j(t) := e_j - w_j(t)\) and keep the release dates and processing times unchanged. Then the corollary follows from Theorem \(\ref{7}\) and Theorem \(\ref{8}\) \(\square\)

Observe that the objective function \(\sum S f_j(C_j) + \sum S e_j\) is in particular a generalization of weighted flow time with rejection \((\sum S w_j(C_j - r_j) + \sum S e_j\) where each job \(j\) has a weight \(w_j\) associated with it), weighted tardiness with rejection \((\sum S w_j \max\{0, t - d_j\} + \sum S e_j\) and each job \(j\) has additionally a deadline \(d_j\), and weighted sum of completion times with rejection \((\sum S w_j C_j + \sum S e_j)\). So our algorithm works for these objective functions as well.

When only the number \(s\) of scheduled jobs is chosen as parameter the problem becomes \(W[1]\)-hard, as pointed out to us by an anonymous reviewer.

**Theorem 9.** Scheduling a set \(J\) of \(s\) jobs from among \(n\) given jobs (non-) preemptively on a single machine to maximize \(\sum_{j \in J} b_j - w_j(C_j)\) is \(W[1]\)-hard for parameter \(s\), even if all \(n\) jobs have the same function \(w_j\) and \(b_j = p_j\) for each job \(j\). Thus, (non-)preemptive \(s\)-\text{GPSP} is \(W[1]\)-hard for parameter \(s\).

**Proof.** Consider an instance of \(k\)-\text{SUBSET SUM}, specified by integers \(s_i\) and a target value \(q\). In our reduction, for each \(s_i\), we create a job \(j\) with processing time \(p_j := s_j\), profit \(b_j := s_j\) and cost function \(w_j\) defined as

\[
w_j(t) = \begin{cases} 
0, & \text{if } t \leq q, \\
\infty, & \text{if } t > q.
\end{cases}
\]

for all times \(t\). Then there is a set of \(s\) integers \(s_i\) whose sum is exactly \(q\), if and only if we can schedule the corresponding set \(J\) of \(s\) jobs preemptively or non-preemptively on a single machine such that \(\sum_{j \in J} b_j - w_j(C_j) = q\). Notice that there is only a single function \(w_j(t)\) that is used for all jobs \(j \in J\) in this reduction.

When defining as profit function \(f_j(t) := b_j - w_j(t)\) for each job \(j\) this yields an instance of \(k\)-\text{GPSP} which is thus \(W[1]\)-hard. \(\square\)
4.3 Hardness for Constantly Many Processing Times

On the other hand, we prove that choosing only the maximum processing time \( p_{\text{max}} \), or number of distinct processing times \( \bar{p} \) as a parameter is not enough, as we show the problem to be \( \text{NP-hard} \) even if \( p_j \in \{1, 3\} \) for all jobs \( j \). The same holds for the General Scheduling Problem (GSP) where—without the profit maximization aspect—we are given a set of jobs \( j \), each of them having a release date \( r_j \) and an individual cost function \( w_j(t) \), and we want to schedule all of them on one machine in order to minimize \( \sum_j w_j(C_j) \) where \( C_j \) denotes the completion time of job \( j \) in the computed (preemptive) schedule.

**Theorem 10.** The General Profit Scheduling Problem (GPSP) and the General Scheduling Problem (GSP) are \( \text{NP-hard} \), even if the processing time of each job is exactly 1 or 3. This holds in both the preemptive and non-preemptive setting.

We now show that GPSP is \( \text{NP-hard} \), even if \( p_j \in \{1, 3\} \) for each job \( j \). We reduce from \textbf{3-DIMENSIONAL MATCHING}, where one is given three sets \( A = \{a_1, \ldots, a_n\}, B = \{b_1, \ldots, b_n\}, \) and \( C = \{c_1, \ldots, c_n\} \) for some \( n \in \mathbb{N} \), and a set \( T \subseteq A \times B \times C \). The goal is to find a subset \( T' \subseteq T \) with \( |T'| = n \) such that for any two different triples \( (a_i, b_j, c_k), (a'_i, b'_j, c'_k) \in T' \) it holds that \( a_i \neq a'_i, b_j \neq b'_j, \) and \( c_k \neq c'_k \). Note that this implies that each element in \( A, B, \) and \( C \) appears exactly once in \( T' \). We say that the given instance is a “yes”-instance if there exists such a set \( T' \). The \textbf{3-DIMENSIONAL MATCHING} problem is \( \text{NP-hard} \).

Given an instance of \textbf{3-DIMENSIONAL MATCHING}, we construct an instance of GPSP. With each triple \( (a_i, b_j, c_k) \in T \) we associate an interval of length 3 on the time axis. For each such triple, we define a value \( t(a_i, b_j, c_k) \) such that for any two different triples, the corresponding intervals \( [t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3) \) are disjoint and \( \bigcup_{(a_i, b_j, c_k) \in T} (t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3) = [0, 3|T|] \). This can for instance be achieved by assigning to each triple \( (a_i, b_j, c_k) \in T \) a unique identifier \( z(a_i, b_j, c_k) \in \{0, |T| - 1\} \) and defining \( t(a_i, b_j, c_k) := 3 \cdot z(a_i, b_j, c_k) \). For each triple \((a_i, b_j, c_k) \in T\), we introduce four jobs, all released at time \( t(a_i, b_j, c_k) \): three jobs of unit length called \( A(a_i, b_j, c_k), B(a_i, b_j, c_k), C(a_i, b_j, c_k) \) that represent the occurrence of the respective element from \( A, B, C \) in the triple \((a_i, b_j, c_k)\) (so the occurrence of the elements \( a_i, b_j, \) and \( c_k \) in \((a_i, b_j, c_k)\), respectively); and additionally one job of length three called \( L(a_i, b_j, c_k) \).

For each element in \( A \cup B \cup C \) we specify a unique point in time. We define \( t(a_i) := 3|T| + i, t(b_j) := 3|T| + n + j, \) and \( t(c_k) := 3|T| + 2n + k \) for each respective element \( a_i \in A, b_j \in B, c_k \in C \); see Figure 1 for a sketch of the subdivision of the time axis and the profit functions (that we define later).

Before specifying the profit function for each job, we give some intuition. The idea is that if the given instance is a “yes”-instance, then there is an optimal schedule where

- for each triple \((a_i, b_j, c_k)\) in the optimal solution we schedule the job \( L(a_i, b_j, c_k) \) during the interval \([t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3) \) and the jobs \( A(a_i, b_j, c_k), B(a_i, b_j, c_k), \) and \( C(a_i, b_j, c_k) \) during interval \([t(a_i) - 1, t(a_i)), [t(b_j) - 1, t(b_j)], \) and \([t(c_k) - 1, t(c_k)]\), respectively.

- for each triple \((a_i, b_j, c_k)\) that is not in the optimal solution we schedule jobs \( A(a_i, b_j, c_k), B(a_i, b_j, c_k), C(a_i, b_j, c_k) \) during \([t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3) \) (in an arbitrary order) and the job \( L(a_i, b_j, c_k) \) somewhere after time \( 3|T| + 3n \).

For defining the profit functions, let \( M_1 := 1 + 2^{3n} \) and \( M_2 := M_1 \cdot 3|T| + 2^{3n} \). For the jobs \( A(a_i, b_j, c_k), B(a_i, b_j, c_k), \) and \( C(a_i, b_j, c_k) \) for a triple \((a_i, b_j, c_k) \in T\) we define their profit function as follows:

\[
\begin{align*}
    f_{A(a_i, b_j, c_k)}(t) &:= \begin{cases} 
    M_1 + M_2, & \text{for } t \leq t(a_i, b_j, c_k) + 3, \\
    2^{3n-i} + M_2, & \text{for } t(a_i, b_j, c_k) + 3 < t \leq t(a_i), \\
    0, & \text{for } t > t(a_i);
    \end{cases} \\
    f_{B(a_i, b_j, c_k)}(t) &:= \begin{cases} 
    M_1 + M_2, & \text{for } t \leq t(a_i, b_j, c_k) + 3, \\
    2^{2n-j} + M_2, & \text{for } t(a_i, b_j, c_k) + 3 < t \leq t(b_j), \\
    0, & \text{for } t > t(b_j);
    \end{cases}
\end{align*}
\]
3 \cdot M_1 + 3 \cdot M_2

3 \cdot M_1 + 3 \cdot M_2

M_1 + M_2

2^{3n-i} + M_2

2^{2n-j} + M_2

2^{n-k} + M_2

3 \cdot M_1 + 3 \cdot M_2

3 \cdot M_1 + 3 \cdot M_2

\text{Fig. 1. Sketch for the reduction in Theorem 10}

f_{C(a_i, b_j, c_k)}(t) := \begin{cases} M_1 + M_2, & \text{for } t \leq t(a_i, b_j, c_k) + 3, \\ 2^{n-k} + M_2, & \text{for } t(a_i, b_j, c_k) + 3 < t \leq t(c_k), \\ 0, & \text{for } t > t(c_k). \end{cases}

For the long job L(a_i, b_j, c_k) for a triple (a_i, b_j, c_k) \in T we define its profit function as

f_{L(a_i, b_j, c_k)}(t) := \begin{cases} 3 \cdot M_1 + 3 \cdot M_2, & \text{for } t \leq t(a_i, b_j, c_k) + 3, \\ 3 \cdot M_2, & \text{for } t > t(a_i, b_j, c_k) + 3. \end{cases}

To show the correctness of our reduction, we show that there is a solution with overall profit of |T| \cdot (3 \cdot M_1 + 3 \cdot M_2) + |T| \cdot 3 \cdot M_2 + 2^{3n} - 1 if and only if the instance of 3-DIMENSIONAL MATCHING is a “yes”-instance.

Lemma 9. If the given instance of 3-DIMENSIONAL MATCHING is a “yes”-instance, then there is a solution with profit |T| \cdot (3 \cdot M_1 + 3 \cdot M_2) + |T| \cdot 3 \cdot M_2 + 2^{3n} - 1.

Proof. Let T' \subseteq T be a solution of the instance of 3-DIMENSIONAL MATCHING (note that |T'| = n). For each tuple (a_i, b_j, c_k) \in T' we schedule the job L(a_i, b_j, c_k) during [t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3], and the jobs A(a_i, b_j, c_k), B(a_i, b_j, c_k), C(a_i, b_j, c_k) during [t(a_i) - 1, t(a_i)), [t(b_j) - 1, t(b_j)), and [t(c_k) - 1, t(c_k)), respectively. Note that since T' is a feasible solution, at most one job is scheduled in each interval [t(a_i) - 1, t(a_i)), [t(b_j) - 1, t(b_j)), [t(c_k) - 1, t(c_k)]. For all tuples (a_i, b_j, c_k) \notin T' we schedule the job L(a_i, b_j, c_k) in some arbitrary interval during [3|T| + 3n, \infty) and the jobs A(a_i, b_j, c_k), B(a_i, b_j, c_k), and C(a_i, b_j, c_k) during [t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3]. This solution yields an overall profit of 3|T| \cdot (M_1 + M_2) + \sum_{\ell=0}^{3n-1}(M_2 + 2^\ell) + (|T| - n) \cdot 3 \cdot M_2 = |T| \cdot (3 \cdot M_1 + 3 \cdot M_2) + |T| \cdot 3 \cdot M_2 + 2^{3n} - 1. \qed
For the converse of Lemma \[9\], we first show that there is always an optimal solution to the defined GPSP instance that is in standard form. We say that a solution is in standard form if

1. each job \(A(a_i, b_j, c_k)\) (job \(B(a_i, b_j, c_k)\), job \(C(a_i, b_j, c_k)\)) is scheduled either during interval \([t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3]\) or during \([3|T|, t(a_i)]\) (during \([3|T|, t(b_j)]\), during \([3|T|, t(c_k)])\); and
2. each job \(L(a_i, b_j, c_k)\) is scheduled non-preemptively either completely during interval \([t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3]\) or completely during \([3|T| + 3n, \infty]\).

Observe in particular that solutions in standard form schedule all jobs.

**Lemma 10.** There is always an optimal solution of the instance of GPSP that is in standard form.

**Proof.** Given an optimal solution to the defined GPSP instance, suppose that it does not satisfy the first property. First, suppose for contradiction that there is a job \(A(a_i, b_j, c_k)\) that finishes after time \(t(a_i)\), thus contributing zero towards the objective. Then the objective value can be at most \(3|T| \cdot (M_1 + M_2) + |T| \cdot (3 \cdot M_1 + 3 \cdot M_2) - M_1 - M_2\) (assuming that all other jobs give the maximal possible profit). However, then there is a strictly better solution \(S'\) which is given by

- scheduling each job \(A(a_i, b_j, c_k)\), \(B(a_i, b_j, c_k)\), and \(C(a_i, b_j, c_k)\) for each triple \((a_i, b_j, c_k) \in T\) during \([t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3]\), respectively, and
- scheduling each job \(L(a_i, b_j, c_k)\) for each triple \((a_i, b_j, c_k) \in T\) somewhere during \([3|T| + 3n, \infty]\).

The solution \(S'\) yields a profit of at least \(3|T| \cdot (M_1 + M_2) + |T| \cdot (3 \cdot M_2) > 3|T| \cdot (M_1 + M_2) + |T| \cdot (3 \cdot M_1 + 3 \cdot M_2) - M_2\), using that \(M_2 > |T| \cdot M_1\). Hence, the first considered schedule was not optimal. For jobs \(B(a_i, b_j, c_k)\) and \(C(a_i, b_j, c_k)\), the claim can be shown similarly.

Secondly, for contradiction suppose that there is a job \(A(a_i, b_j, c_k)\) that is scheduled during interval \([t(a_i, b_j, c_k) + 3, 3\cdot|T|]\). Then the computed schedule yields a profit of at most \(3|T| \cdot (M_1 + M_2) - M_1 + \sum_{i=0}^{3n-1} 2^i + |T| \cdot 3M_2 < 3|T| \cdot (M_1 + M_2) + |T| \cdot 3M_2\), using that \(M_1 > \sum_{i=0}^{3n-1} 2^i = 2^{3n} - 1\). Hence, solution \(S'\) (as defined above) is better than the considered optimal solution, which yields a contradiction. Hence, any optimal solution satisfies the first property.

Suppose now that we are given an optimal solution that violates the second property. Then there is a job \(L(a_i, b_j, c_k)\) that is not completely scheduled during \([t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3]\). Then its contribution for the global profit is \(3 \cdot M_2\) and this does not depend on where exactly after \(t(a_i, b_j, c_k) + 3\) it finishes. Thus, we can move \(L(a_i, b_j, c_k)\) so that it is completely scheduled non-preemptively during \([3|T| + 3n, \infty]\), without changing its contribution to the profit.

\[\square\]

**Lemma 11.** If there is an optimal solution of the GPSP instance yielding a profit of at least \(|T| \cdot (3 \cdot M_1 + 3 \cdot M_2) + |T| \cdot 3 \cdot M_2 + 2^{3n} - 1\), then the instance of 3-DIMENSIONAL MATCHING is a “yes”-instance.

**Proof.** Given a solution to our defined GPSP instance, by Lemma \[10\] we can assume it is in standard form. Let \(T' \subseteq T\) be the set of triples \((a_i, b_j, c_k) \in T\) corresponding to the jobs \(L(a_i, b_j, c_k)\) that are scheduled during their respective intervals \([t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3]\). We claim that they form a feasible solution and that \(|T'| = n\). The profit obtained by the jobs \(L(a_i, b_j, c_k)\) corresponding to the triples in \(T'\) equals \(|T'| \cdot (3 \cdot M_1 + 3 \cdot M_2)\). The total profit given by all other jobs \(L(a_i, b_j, c_k)\) equals \((|T| - |T'|) \cdot 3 \cdot M_2\). For jobs \(A(a_i, b_j, c_k)\), \(B(a_i, b_j, c_k), C(a_i, b_j, c_k)\) such that \(L(a_i, b_j, c_k)\) is not scheduled during \([t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3]\) we can assume that the former are all scheduled during \([t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3]\) (since there they yield the maximum profit and no other job is scheduled there). The profit of those jobs equals then \((|T| - |T'|) \cdot 3(M_1 + M_2)\). Denote by \(J'\) the set of all other jobs \(A(a_i, b_j, c_k)\), \(B(a_i, b_j, c_k), C(a_i, b_j, c_k)\) (i.e., such that \(L(a_i, b_j, c_k)\) is scheduled during \([t(a_i, b_j, c_k), t(a_i, b_j, c_k) + 3]\)). Their total profit equals \(3|T'| \cdot M_2 + \sum_{(a_i, b_j, c_k) \in J'} 2^{3n-i} + \sum_{B(a_i, b_j, c_k) \in J'} 2^{2n-j} + \sum_{C(a_i, b_j, c_k) \in J'} 2^{n-k}\). Thus, the total profit of the solution equals

\[
|T'| \cdot (3 \cdot M_1 + 3 \cdot M_2) + (|T| - |T'|) \cdot 3 \cdot M_2 + (|T| - |T'|) \cdot 3(M_1 + M_2) + 3|T'| \cdot M_2
+ \sum_{A(a_i, b_j, c_k) \in J'} 2^{3n-i} + \sum_{B(a_i, b_j, c_k) \in J'} 2^{2n-j} + \sum_{C(a_i, b_j, c_k) \in J'} 2^{n-k}
\]

\[= |T| \cdot (3 \cdot M_1 + 3 \cdot M_2) + |T| \cdot 3 \cdot M_2 + \sum_{A(a_i, b_j, c_k) \in J'} 2^{3n-i} + \sum_{B(a_i, b_j, c_k) \in J'} 2^{2n-j} + \sum_{C(a_i, b_j, c_k) \in J'} 2^{n-k}\]

18
Thus, if the total profit is at least $|T| \cdot (3 \cdot M_1 + 3 \cdot M_2) + |T| \cdot 3 \cdot M_2 + 2^{3n} - 1$ then it must be that
\[ \sum_{A(a_i, b_j, c_k) \in J'} 2^{3i-1} + \sum_{B(a_i, b_j, c_k) \in J'} 2^{2j-2} + \sum_{C(a_i, b_j, c_k) \in J'} 2^{n-k} \geq 2^{3n} - 1. \]
Therefore, there must be at least one (and by construction also at most one) one job $A(a_i, b_j, c_k) \in J'$ with $i = 1$, and similarly for each $\ell \in \{1, \ldots, n\}$, there are at least $\ell$ jobs $A(a_i, b_j, c_k) \in J'$ with $i \leq \ell$. This implies that
\[ \sum_{\ell=1}^{2^{3n}} 2^{2n-\ell} = 2^{3n} - 1 - \sum_{\ell=1}^{2n-1} 2^{\ell} = 2^{3n} - 1 - 2^{2n} + 1 = 2^{3n} - 2^{2n} \text{ and thus } \sum_{B(a_i, b_j, c_k) \in J'} 2^{2n-j} + \sum_{C(a_i, b_j, c_k) \in J'} 2^{n-k} \geq 2^{2n} - 1. \]

By the same argument we can show that for each $\ell \in \{1, \ldots, n\}$ there are at least $\ell$ jobs $B(a_i, b_j, c_k) \in J'$ with $j \leq \ell$, and then that for each $\ell \in \{1, \ldots, n\}$ there are at least $\ell$ jobs $C(a_i, b_j, c_k) \in J'$ with $k \leq \ell$. Altogether, this implies that for each $i \in \{1, \ldots, n\}$ there is exactly one job $A(a_i, b_j, c_k) \in J'$, and similarly for each $j \in \{1, \ldots, n\}$ and each $k \in \{1, \ldots, n\}$ there is exactly one job $B(a_i, b_j, c_k) \in J'$ and one job $C(a_i, b_j, c_k) \in J'$. Thus, the set $T'$ is a feasible solution, and in particular $|T'| = n$. \hfill \qed

Lemmas 9 and 11 together show the correctness of the above reduction, and hence complete the proof of Theorem 10. Since by Lemma 10 we can restrict ourselves to solutions in standard form and those are non-preemptive, our reduction works for the preemptive as well as for the non-preemptive setting.

Using the fact that solutions in standard form schedule all jobs, we can modify the above transformation to show that also the General Scheduling Problem (GSP) is NP-hard if the processing time of each job is either exactly 1 or 3. All we need to do is to define a cost function $w_j$ for each job by setting $w_j(t) := 3 \cdot M_1 + 3 \cdot M_2 - f_j(t)$, which in particular ensures that for no completion time any job has negative costs. Then the goal is to find a feasible schedule on one machine to minimize $\sum w_j(C_j)$.

**Corollary 4.** The General Scheduling Problem (GSP) is NP-hard, even if the processing time of each job is either exactly 1 or 3. This holds in the preemptive setting as well as in the non-preemptive setting.

### 5 Discussion and Future Directions

In this paper, we obtained the first fixed-parameter algorithms for several fundamental problems in scheduling. We identified key problem parameters that determine the complexity of an instance, so that when these parameters are fixed the problem becomes polynomial time solvable.

There are several interesting open questions in the connection of scheduling and FPT. One important scheduling objective is weighted flow time (which in the approximative sense is not completely understood yet, even on one machine). It would be interesting to investigate whether this problem is FPT when parameterized by, for instance, an upper bound on the maximum processing time and the maximum weight of a job (assuming integral input data). For scheduling jobs non-preemptively on identical machines to minimize the makespan, in Section 2 we proved the first FPT result for this problem. It would be interesting to extend this to the setting with release dates or to consider as parameter the number of distinct processing times (rather than an upper bound on those). When the jobs have precedence constraints, a canonical parameter would be the width of the partial order given by these constraints, and it would be interesting to study whether $P|\prec\prec|C_{\text{max}}$ admits FPT algorithms when parameterized by the partial order width and additionally by the maximum processing time. For the basic setting of makespan minimization $P||C_{\text{max}}$, only recently Goemans and Rothvöß [30] managed to give an algorithm of running time $O(n^m)$ for all numbers $m$ of machines—thus, it seems rather challenging to obtain a fixed-parameter algorithm parameterized by $m$. For scheduling with rejection, we further suggest the number of distinct rejection costs as a parameter to investigate. Finally, it would be desirable to understand the kernelizability of the scheduling problems we considered here.

**Acknowledgment.** We thank an anonymous reviewer of an earlier version for suggestions how to remove one parameter in the algorithms in Sect. 4 and to prove Theorem 9.
References

1. Marx, D.: Fixed-parameter tractable scheduling problems. In: Packing and Scheduling Algorithms for Information and Communication Services (Dagstuhl Seminar 11091). Volume 1. (2011) 86
2. Afrati, F., Bampis, E., Chekuri, C., Karger, D., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., Sviridenko, M.: Approximation schemes for minimizing average weighted completion time with release dates. In: Proc. FOCS 1999. (1999) 32–43
3. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: Theoretical and practical results. J. ACM 34 (1987) 144–162
4. Fellows, M.R., Gaspers, S., Rosamond, F.A.: Parameterizing by the number of numbers. Theory Comput. Syst. 50(4) (2012) 675–693
5. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. 46(1-3) (1990) 259–271
6. Bansal, N., Pruhs, K.: The geometry of scheduling. In: Proc. FOCS 2010. (2010) 407–414
7. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. 17 (1969) 263–269
8. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: An application of bin-packing to multiprocessor scheduling. SIAM J. Comput. 7 (1978) 1–17
9. Friesen, D.K.: Tighter bounds for the multifit processor scheduling algorithm. SIAM J. Comput. 13 (1984) 170–181
10. Langston, M.A.: Processor scheduling with improved heuristic algorithms. PhD thesis, Texas A&M University (1981)
11. Sahni, S.: Algorithms for scheduling independent tasks. J. ACM 23 (1976) 116–127
12. Ebenlendr, T., Krčál, M., Sgall, J.: Graph balancing: a special case of scheduling unrelated parallel machines. In: Proc. SODA 2008. (2008) 483–490
13. Shmoys, D.B., Tardos, É.: An approximation algorithm for the generalized assignment problem. Math. Program. 62(1-3) (1993) 461–474
14. Svensson, O.: Santa claus schedules jobs on unrelated machines. SIAM J. Comput. 41(5) (2012) 1318–1341
15. Bansal, N., Dhamdhere, K.: Minimizing weighted flow time. ACM Trans. Algorithms 3(4) (November 2007)
16. Chekuri, C., Khanna, S., Zhu, A.: Algorithms for minimizing weighted flow time. In: Proc. STOC 2001. (2001) 84–93
17. Chekuri, C., Khanna, S.: Approximation schemes for preemptive weighted flow time. In: Proc. STOC 2002. (2002) 297–305
18. Engels, D.W., Karger, D.R., Kolliopoulos, S.G., Sengupta, S., Uma, R.N., Wein, J.: Techniques for scheduling with rejection. J. Algorithms 49(1) (2003) 175–191
19. Sviridenko, M., Wiese, A.: Approximating the configuration-LP for minimizing weighted sum of completion times on unrelated machines. In: Proc. IPCO 2013. Volume 7801 of Lecture Notes Comput. Sci. (2013) 387–398
20. Hoogeveen, H., Skutella, M., Woeginger, G.J.: Preemptive scheduling with rejection. Math. Program. 94(2-3, Ser. B) (2003) 361–374
21. van Bevern, R., Mnich, M., Niedermeier, R., Weller, M.: Interval scheduling and colorful independent sets. In: Algorithms and Computation. Volume 7676 of Lecture Notes Comput. Sci. (2012) 247–256
22. Bodlaender, H.L., Fellows, M.R.: W[2]-hardness of precedence constrained K-processor scheduling. Oper. Res. Lett. 18(2) (1995) 93–97
23. Fellows, M.R., McCartin, C.: On the parametric complexity of schedules to minimize tardy tasks. Theoret. Comput. Sci. 298(2) (2003) 317 – 324
24. Marx, D., Schlotter, I.: Stable assignment with couples: Parameterized complexity and local search. Discr. Optimization 8(1) (February 2011) 25–40
25. Chu, G., Gaspers, S., Narodytska, N., Schutt, A., Walsh, T.: On the complexity of global scheduling constraints under structural restrictions. In: Proc. IJCAI 2013. (2013)
26. Lenstra, Jr., H.W.: Integer programming with a fixed number of variables. Math. Oper. Res. 8(4) (1983) 538–548
27. Khachiyan, L.G.: Convexity and complexity in polynomial programming. In: Proc. Internat. Congress of Mathematicians, Vol. 1, 2. (1984) 1569–1577
28. Heinz, S.: Complexity of integer quasiconvex polynomial optimization. J. Complexity 21(4) (2005) 543–556
29. Köppe, M.: On the complexity of nonlinear mixed-integer optimization. In Lee, J., Leyffer, S., eds.: Mixed Integer Nonlinear Programming. Volume 154 of The IMA Volumes in Mathematics and its Applications. (2012) 533–557
30. Goemans, M.X., Rothvoß, T.: Polynomiality for bin packing with a constant number of item types. In: Proc. SODA 2014. (2014) To appear.
31. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer, Berlin (2003)
32. Jansen, K., Kratsch, S., Marx, D., Schlotter, I.: Bin packing with fixed number of bins revisited. J. Comput. System Sci. 79(1) (2013) 39–49
33. Smith, W.E.: Various optimizers for single-stage production. Naval Res. Logist. Quart. 3 (1956) 59–66
34. Chudak, F.A., Hochbaum, D.S.: A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. Oper. Res. Lett. 25(5) (1999) 199–204
35. Lenstra, J., Kan, A.R., Brucker, P.: Complexity of machine scheduling problems. In: Studies in Integer Programming. Volume 1 of Ann. Discrete Math. (1977) 343–362
36. Fellows, M.R., Koblitz, N.: Fixed-parameter complexity and cryptography. In: Applied algebra, algebraic algorithms and error-correcting codes (San Juan, PR, 1993). Volume 673 of Lecture Notes Comput. Sci. (1993) 121–131
37. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. ACM 42(4) (July 1995) 844–856
38. Garey, M.R., Johnson, D.S.: Computers and intractability. W. H. Freeman and Co., San Francisco, Calif. (1979)