Single Machine Scheduling with Few Deadlines

Klaus Heeger
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Danny Hermelin
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Dvir Shabtay
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

We study single-machine scheduling problems with few deadlines. We focus on two classical objectives, namely minimizing the weighted number of tardy jobs and the total weighted completion time. For both problems, we give a pseudopolynomial-time algorithm for a constant number of different deadlines. This algorithm is complemented with an ETH-based, almost tight lower bound. Furthermore, we study the case where the number of jobs with a nontrivial deadline is taken as parameter. For this case, the complexity of our two problems differ: Minimizing the total number of tardy jobs becomes fixed-parameter tractable, while minimizing the total weighted completion time is W[1]-hard.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → W hierarchy; Theory of computation → Dynamic programming; Theory of computation → Scheduling algorithms

Keywords and phrases Single-machine scheduling, weighted completion time, tardy jobs, pseudo-polynomial algorithms, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.24

Funding Supported by the ISF, grant No. 1070/20.

1 Introduction

Already since the 1950s, scheduling has been an important area of combinatorial optimization, with various applications coming from a broad range of areas such as manufacturing, management, and healthcare [2, 19]. This lead to a wide variety of different scheduling problems, depending on the targeted applications. What almost all scheduling problems have in common is that there is a set of jobs $\{1, \dots, n\}$ with different characteristics that need to be processed on one or several machines, subject to some feasibility constraints, and with the objective of optimizing a predefined objective function. Usually, the objective function is based on the completion times of the jobs in the proposed solution schedule.

One very basic scheduling setting is the following: We are given a set of n jobs, all available to be non-preemptively processed on a single machine at time zero. Each job j has a processing time p_j, a weight w_j (corresponding to its importance), and a due date d_j. The jobs must be processed one after another on a single machine. In this setting, the *completion time* C_j of a job j is simply the sum of processing times of all jobs scheduled before j (including j itself). A job is *tardy* if its completion time is larger than its due date. The tardiness of a job can also be expressed using the *unit-penalty* function U_j which is one if job j is tardy and zero otherwise.

Two of the most-common objective functions in single-machine scheduling are the total weighted completion time $\sum w_j C_j$, and the total weighted number of tardy jobs $\sum w_j U_j$. In the classic 3-field notation by Graham *et al.* [8], these two problems are denoted by $1||\sum w_j C_j$ and $1||\sum w_j U_j$.

A less well-studied generalization of these two problems is when the jobs also have deadlines $\overline{d}_1, \ldots, \overline{d}_n$. Deadlines differ from due dates in that they must be met, and so scheduling problems with deadlines require solution schedules to have all jobs complete before their deadline. Adding job deadlines to $1||\sum w_j C_j$ and $1||\sum w_j U_j$ results in problems $1|\overline{d}_j|\sum w_j C_j$ and $1|\overline{d}_j|\sum w_j U_j$. Both these problems have natural applications in practice: In the $1|\overline{d}_j|\sum w_j C_j$ problem we wish to minimize the total completion time (or equivalently average completion time) under the additional deadline constraints, while in $1|\overline{d}_j|\sum w_j U_j$ we can interpret the weight of a job as a premium for early completion of the job (before its due date instead of before its deadline) [9].

Note that the two problems above have quite efficient solutions when the jobs have no deadlines. The $1||\sum w_j C_j$ problem can be solved in $O(n \log n)$ time [22] by scheduling all jobs according to the weighted shortest processing time (WSPT) order. The $1||\sum w_j U_j$ problem is weakly NP-hard [14], but it admits an $O(Pn)$ pseudo-polynomial time algorithm [16], where $P = \sum p_j$ is the total sum of processing times of all input jobs, and an $O(n \log n)$ time algorithm [18] when all jobs have unit weight (*i.e.* $w_j = 1$ for any job j). Thus, it is natural to ask whether these results can be generalized to the case where there are only a few different deadlines among all n jobs, or when only a few jobs have nontrivial deadlines (*i.e.*, jobs j with deadline $\overline{d}_j < P$). This question is the starting point of this paper.

Our Contribution. We investigate the parameterized complexity of $1|\overline{d}_j|\sum w_j U_j$ and $1|\overline{d}_j|\sum w_j C_j$ with respect to two different parameters: The first is the number of different deadlines among all n jobs, and the second is the number of jobs with a nontrivial deadline, *i.e.* the number of jobs j with $\overline{d}_j < P$. Note that the latter parameter upper bounds the former. For the number of different deadlines parameter, both problems behave similarly: They are weakly NP-hard already for two different deadlines (one of which is a "trivial" deadline equaling the total processing time). When considering the case where all numbers (processing times and weights) are encoded in unary, both problems are W[1]-hard but admit XP-time algorithms.

For the number of jobs with nontrivial deadline parameter, the complexity of both problems diverge: For $1|\overline{d}_j|\sum w_j C_j$, the hardness results for the number of different deadlines carry over. For $1|\overline{d}_j|\sum w_j U_j$, however, there is an $O(2^k \cdot P \cdot n)$-time algorithm. For the special case of unit weights (*i.e.* $w_j = 1$ for all j), this algorithm can be improved to an $O(2^k \cdot n \log n)$-time algorithm. We refer to Table 1 for an overview of our results.

Table 1 Overview of our results. We use n to denote the number of jobs, and P to denote the total processing time of all jobs.

	k =# different deadlines	k =# jobs with deadline
$1\|\overline{d}_j\|\sum w_j U_j$	weakly NP-hard even if $k = 2$ and $w_j = 1$ (3.1) W[1]-hard with unary input even if $w_j = 1$ (3.1) $P^{O(k)}$-time algorithm (3.4)	weakly NP-hard if $k = 0$ [14] $O(2^k n \log n)$ if $w_j = 1$ (3.6) $O(2^k Pn)$ (3.5)
$1\|\overline{d}_j\|\sum w_j C_j$	weakly NP-hard even if $k = 2$ [17] W[1]-hard even with unary input (4.2) $P^{O(k)}$-time algorithm (4.4)	

Related Work. While there are numerous papers on single-machine scheduling, rather few consider deadlines. Regarding the first problem we study, $1|\overline{d}_j|\sum w_jU_j$, Lawler [15] showed that $1|\overline{d}_j|\sum U_j$ (the unit weight special case of $1|\overline{d}_j|\sum w_jU_j$) is weakly NP-hard. This result was strengthened by Yuan [24] who showed that $1|\overline{d}_j|\sum U_j$ is strongly NP-hard. Huo *et al.* [12] showed that $1|\overline{d}_j|\sum U_j$ can be solved in polynomial time if either $d_i \leq d_j$ implies $\overline{d}_i \leq \overline{d}_j$ and $p_i \leq p_j$, or if $p_i \leq p_j$ implies $d_i - p_i \geq d_j - p_j$. The running times of some of these algorithms were improved later [11]. The problem $1|\overline{d}_j|\sum w_jU_j$ was also studied from a practical point of view: Hariri and Potts [9] designed a branch-and-bound based algorithm for it, while Baptiste *et al.* [1] designed an ILP-formulation with n variables and $2n$ constraints for the problem.

Concerning the second problem we study, $1|\overline{d_j}|\sum w_jC_j$, Lenstra *et al.* [17] proved that the problem is weakly NP-hard even if only one job has a nontrivial deadline. A related problem is $1|r_j, \overline{d}_j, \text{pmtn}|\sum C_j$. Here, the jobs additionally have release dates (that is, a job cannot be scheduled before its release date) and preemption is allowed (that is, it is possibly to process a job j partially, then schedule other jobs, and later continue processing job j). Further, all jobs have weight 1. Wan *et al.* [23] showed weak NP-hardness of $1|r_j, \overline{d}_j, \text{pmtn}|\sum C_j$. Recently, strong NP-hardness was shown [4]. In case of "agreeable" processing times and deadlines (that is, whenever the deadline of job j is larger than the one of job j', then also the processing time of j is larger than the processing time of j'), the problem is solvable in polynomial time [10].

Chen and Yuan [3] studied the problem of minimizing the total tardiness $\sum T_j$ on a single machine with deadlines (the tardiness of a job j is $T_j = \max\{0, C_j - d_j\}$). Chen and Yuan [3] showed that this problem (denoted $1|\overline{d}_j|\sum T_j$) is strongly NP-hard. A similar problem is that of minimizing the total late work $\sum Y_j$ of all jobs, where the late work of a job j is $Y_j = \min\{p_j, T_j\}$. Chen *et al.* [5] showed $1|\overline{d}_j|\sum Y_j$ is strongly NP-hard. However, if all jobs share a common due date, then the problem is weakly NP-hard and admits a pseudopolynomial-time algorithm. Chen *et al.* also showed that a few special cases of the problem become polynomial-time solvable.

2 Preliminaries

We consider non-preemptive scheduling problems on a single machine. Here, the input consists of n jobs $\{1, \ldots, n\}$ which are all available to be processed at time zero. We denote by $[n] := \{1, 2, \ldots, n\}$. Each job j is characterized by its *processing time* $p_j \in \mathbb{N}$, its *weight* $w_j \in \mathbb{N}$, and its *deadline* $\overline{d}_j \in \mathbb{N}$. In one of the problems we consider below, each job j will also have a *due date* $d_j \in \mathbb{N}$. We assume without loss of generality that $d_j \leq \overline{d}_j$ holds for every job $j \in [n]$. We denote the total processing time of all jobs by $P = \sum_{j\in[n]} p_j$, and their total weight by $W = \sum_{j\in[n]} w_j$.

A *schedule* is a permutation $\sigma : [n] \to [n]$ of the jobs. Given a schedule σ, the *completion time* $C_j(\sigma)$ of job j is $C_j(\sigma) := \sum_{i\in[n]:\sigma(i)\leq\sigma(j)} p_i$; that is, it is the total processing times of all jobs preceding j in the schedule (including j itself). A schedule σ is *feasible* if $C_j(\sigma) \leq \overline{d}_j$ for all $j \in [n]$. A job j is *early* in σ if $C_j(\sigma) \leq d_j$, and otherwise it is *tardy* in σ. We use $U(\sigma)$ to denote the set of all jobs that are tardy in σ. We call a deadline $\overline{d}_j$ *trivial* if $\overline{d}_j \geq P$ (that is, any schedule fulfills this deadline). We assume without loss of generality that $\overline{d}_j = P$ is the only occurring trivial deadline.

We focus on instances with few different deadlines. Thus, we set $\overline{d}^{(1)}$ to be the smallest appearing deadline, $\overline{d}^{(2)}$ to be the second-smallest appearing deadline (different from $\overline{d}^{(1)}$), and so on. Note that the largest deadline $\overline{d}^{(k)}$ will always be P. For each $i \in [k]$, we denote by $J^{(i)}$ the set of jobs with deadline $\overline{d}^{(i)}$, and by $P^{(i)}$ the total processing time of all jobs in $J^{(i)}$.

We study two problems in this paper that differ according to the objective function used to evaluate feasible schedules:

$1|\overline{d}_j|\sum w_jU_j$
Input: A set of n jobs with due dates, a number b.
Question: Is there a feasible schedule σ such that $\sum_{j\in U(\sigma)} w_j \leq b$?

$1|\overline{d}_j|\sum w_jC_j$
Input: A set of n jobs, a number b.
Question: Is there a feasible schedule σ such that $\sum_{j\in[n]} w_jC_j(\sigma) \leq b$?

The problem names are derived from the classical 3-field notation by Graham *et al.* [8], where the first field encodes the machine setting (in our case, "1" represents a single machine), the second field contains constraints (in our setting, $\overline{d}_j$ indicates the existence of deadlines), and the third field containing the objective function (in our case either the weighted completion time $\sum w_jC_j$ or the weighted number of late jobs $\sum w_jU_j$).

3 The $1|\overline{d}_j|\sum w_jU_j$ problem

In this section, we study the problem of minimizing the weighted number of tardy jobs. We start by showing some hardness results for $1|\overline{d}_j|\sum w_jU_j$ in Section 3.1. Afterwards, we give a pseudopolynomial-time algorithm for constant number of different deadlines in Section 3.2. Finally, in Section 3.3, we design efficient algorithms for the case of few jobs having a nontrivial deadline.

3.1 Hardness results

Recently, Yuan [24] showed that $1|\overline{d}_j|\sum U_j$ is strongly NP-hard via a reduction from 3-Partition. This NP-hard problem is a special case of the following Multiway Number Partitioning partition problem:

Multiway Number Partitioning
Input: Integers m and k, and $t := m \cdot k$ numbers $a_1, \ldots, a_t$.
Question: Is there a partition $(A_1, \ldots, A_k)$ of $\{a_1, \ldots, a_t\}$ such that $|A_i| = m$ for each $i \in [k]$ and $\sum_{a\in A_i} a = \sum_{a\in A_j} a$ for all $i, j \in [k]$?

Observe that Multiway Number Partitioning with $k = 2$ is known as the weakly NP-hard Equal Cardinality Partition problem [7]. Furthermore, the classical Bin Packing problem naturally reduces to Multiway Number Partitioning by adding items of zero size in order to obtain an instance with $t = m \cdot k$ numbers.

Yuan [24] presented a reduction from Multiway Number Partitioning with $m = 3$ to $1|\overline{d}_j|\sum U_j$. His reduction uses in its construction the sum of all input numbers $B := \sum_{\ell=1}^{t} a_\ell$ and a sufficiently large number $M = \frac{3}{2}t(t+1)B + 1$. It creates a job (i, j) for each $i \in [k]$ and $j \in [t]$ with processing time $p_{i,j} = M^2 + i \cdot (M + a_j)$. For each $i \in [k]$, the jobs (i, j) share the same deadline $\overline{d}_{i,j} = \overline{d}^{(i)} = \sum_{\ell=1}^{i-1} P^{(\ell)} + t \cdot \sum_{\ell=1}^{m} P^{(\ell)}$, where

$P^{(\ell)} = 3M^2 + 3\ell M + 3\ell B$. Furthermore, for each $j \in [t]$, the jobs (i, j) share the same due date $d_{i,j} = d^{(j)} = j \cdot M^2 + \frac{3}{2}Mt(t+1)B$. Yuan [24] proved that if the input Multiway Number Partitioning instance $(3, k, a_1, \ldots, a_t)$ is a yes-instance then the constructed job set has a feasible schedule with at most $b = 3t^2 - t$ tardy jobs.

Note that the reduction from Multiway Number Partitioning of Yuan [24] described above works for any value of $m \geq 3$ and $k \geq 2$. Setting $m = t/2$ and $k = 2$ results in a reduction from Equal Cardinality Partition to $1|\overline{d}_j|\sum U_j$ with two different deadlines. Using arbitrary m and k results in a reduction from Bin Packing with k bins to $1|\overline{d}_j|\sum U_j$ with k different deadlines. It is known that Bin Packing with unary encoded numbers is W[1]-hard parameterized by the number k of bins, and assuming ETH it cannot be solved in $f(k) \cdot n^{o(k/\log k)}$ time [13]. Thus, we directly get the following hardness result for $1|\overline{d}_j|\sum U_j$:

▶ **Theorem 3.1.** *Let k denote the number of different deadlines in a $1|\overline{d}_j|\sum U_j$ instance. Then the $1|\overline{d}_j|\sum U_j$ problem is*

- *weakly NP-hard even when $k = 2$,*
- *W[1]-hard with respect to k even when all numbers are encoded in unary, and*
- *admits no $f(k) \cdot P^{o(k/\log k)}$-time algorithm assuming ETH.*

3.2 Constant number of deadlines

Let $k = |\{\overline{d}_j : 1 \leq j \leq n\}|$ denote the number of different deadlines in the job instance. By Theorem 3.1, we know that $1|\overline{d}_j|\sum w_jU_j$ is W[1]-hard with respect to k even if $w_j = 1$ for each job j, and all processing times are encoded in unary. Complementing this result, we will show that if $k = O(1)$, then we can solve $1|\overline{d}_j|\sum w_jU_j$ in pseudo-polynomial time. Throughout the subsection we will assume that the input jobs are ordered by ascending due dates, *i.e.* $d_1 \leq \cdots \leq d_n$. Furthermore, we denote by $J^{(i)}$ the set of jobs with deadline $\overline{d}^{(i)}$ and by $P^{(i)}$ the total processing time of all jobs from $J^{(i)}$.

Our algorithm for $1|\overline{d}_j|\sum w_jU_j$ is based on dynamic programming. Our algorithm processes the jobs from job 1 to n (*i.e.* according to increasing due date), and creates a table τ_j for each $j \in [n]$. The table τ_j is associated with the job set $\{1, \ldots, j\}$, and it contains an entry $\tau_j[x_1, \ldots, x_k]$ for each $(x_1, \ldots, x_k) \in \{0, \ldots, P^{(1)}\} \times \cdots \times \{0, \ldots, P^{(k)}\}$. The entry $\tau_j[x_1, \ldots, x_k]$ shall equal $\upsilon_j[x_1, \ldots, x_k]$, where we define $\upsilon_j[x_1, \ldots, x_k]$ to be the maximum number w^* such that there is a feasible schedule of all n jobs fulfilling that

1. the set S of early jobs from $\{1, \ldots, j\}$ has weight w^*, and
2. for each $i \in [k]$, the total processing time of early jobs from $S \cap J^{(i)}$ equals x_i.

We stress that all considered schedules schedule all n jobs, even if they correspond to some entry $\tau_j[x_1, \ldots, x_k]$ with $j < n$. The minimum value $W - \upsilon_n[x_1, \ldots, x_k]$ over all $(x_1, \ldots, x_k) \in \{0, \ldots, P^{(1)}\} \times \cdots \times \{0, \ldots, P^{(k)}\}$ is the minimum weighted number of tardy jobs of any feasible schedule for our instance. We call a feasible schedule fulfilling the second condition above a $(j; x_1, \ldots, x_k)$-*compatible* schedule.

Our dynamic program needs to be able to compute $\upsilon_j[x_1, \ldots, x_k]$, given $\upsilon_{j-1}[y_1, \ldots, y_k]$ for all $y_1, \ldots, y_k$. There are two cases: First, job j is late in a schedule witnessing the value of $\upsilon_j[x_1, \ldots, x_k]$. In this case, we have $\upsilon_j[x_1, \ldots, x_k] = \upsilon_{j-1}[x_1, \ldots, x_k]$. Second, job j is early in a schedule witnessing the value of $\upsilon_j[x_1, \ldots, x_k]$. In this case, we have $\upsilon_j[x_1, \ldots, x_k] = w_j + \upsilon_{j-1}[x_1, \ldots, x_{i-1}, x_i - p_j, x_{i+1}, \ldots, x_k]$ where $i \in [k]$ such that the deadline of j is $\overline{d}^{(i)}$. However, we do not have $\upsilon_j[x_1, \ldots, x_k] = \max\{\upsilon_{j-1}[x_1, \ldots, x_k], w_j + \upsilon_{j-1}[x_1, \ldots, x_{i-1}, x_i - p_j, x_{i+1}, \ldots, x_k]\}$ because it is not always possible to modify a schedule corresponding to $\upsilon_{j-1}[x_1, \ldots, x_{i-1}, x_i - p_j, x_{i+1}, \ldots, x_k]$ in such a way that also job j is early.

This is the major difficulty in the design of the dynamic program: Finding a criterion where we can modify the schedule corresponding to $\tau_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k]$ so we can additionally schedule job j early.

To derive such a criterion, we first observe that if we knew the set of early jobs (from $[n]$, not only from $[j]$), then we can easily compute an optimal schedule: As we have a strict upper bound for the completion time of each job (either its due date if the job is early, or its deadline if the job is late), it is optimal to schedule the jobs ordered by this strict upper bound.

▶ **Observation 3.2.** *For a given subset $S \subseteq \{1, \dots, n\}$ of jobs, there is a feasible schedule in which each job from S is early if and only if ordering the jobs according to increasing modified due dates*

$$d_j^* = \begin{cases} d_j & \text{if } j \in S \\ \overline{d}_j & \text{if } j \notin S \end{cases}$$

results in a schedule where all jobs are early.

Using Observation 3.2, we now basically know how scheduling job j as well as a given set $S \subseteq [j-1]$ of jobs early looks like: Before j, the early jobs from $[j-1]$ and all jobs whose deadline is smaller than d_j are scheduled. After j, all jobs with deadline larger than d_j (and which are not contained in S) are scheduled, according to increasing deadline. This implies the following criterion on when a schedule witnessing the value of $\upsilon_j[x_1, \dots, x_k]$ can also additionally schedule j early.

▶ **Lemma 3.3.** *Let $j \in [n]$ be a job with deadline $\overline{d}^{(i_1)}$ and $x_1, \dots, x_k \in \{0, 1, \dots, P\}$. Let $i_0 \in [k]$ be minimum such that $d_j \leq \overline{d}^{(i_0)}$. Let σ_{j-1} be a $(j-1; x_1, \dots, x_k)$-compatible schedule where $S \subseteq [j-1]$ is the set of early jobs from $[j-1]$. Then there exists a $(j; x_1, \dots, x_{i_1} - 1, x_{i_1} + p_j, x_{i_1+1}, \dots, x_k)$-compatible schedule σ_j where $S \cup \{j\}$ is early if and only if*

1. *$\sum_{\ell=1}^{i_0-1} P^{(\ell)} + \sum_{\ell=i_0}^{k} x_\ell + p_j \leq d_j$, and*
2. *for each $i \geq i_0$, we have $\sum_{\ell=1}^{i} P^{(\ell)} + \sum_{\ell=i+1}^{k} x_\ell \leq \overline{d}^{(i)}$.*

Proof. ($\Longleftarrow$:) We begin with the reverse direction. Assume that there is a $(j; x_1, \dots, x_{i_1} - 1, x_{i_1} + p_j, x_{i_1+1}, \dots, x_k)$-compatible schedule σ_j where $S \cup \{j\}$ is early. By Observation 3.2, we may assume that σ schedules the jobs in non-decreasing order of modified due dates d_j^* (which are set according to the early set of jobs $S \cup \{j\}$).

We first show Item 1. Before job j, all jobs with deadline smaller than d_j and all early jobs with due date smaller than d_j are scheduled. The processing time of jobs with deadline smaller than d_j is precisely $\sum_{\ell=1}^{i_0-1} P^{(\ell)}$. All jobs from S have due date at most d_j (as we ordered the jobs according to increasing due date). Thus, the total processing time of all early jobs with due date at most d_j but deadline at least d_j equals $p_j + \sum_{\ell=i_0}^{k} x_\ell$. As j is early, we have $\sum_{\ell=1}^{i_0-1} P^{(\ell)} + p_j + \sum_{\ell=i_0}^{k} x_\ell \leq d_j$, *i.e.* Item 1 holds.

We continue by showing Item 2, so fix $i \geq i_0$. The last job with deadline $\overline{d}^{(i)}$ is processed after all jobs with deadline at most $\overline{d}^{(i)}$, as well as all early jobs with due date at most $\overline{d}^{(i)}$. The processing time of jobs with deadline at most $\overline{d}^{(i)}$ is $\sum_{\ell=1}^{i} P^{(\ell)}$. Each job $j' \in S \cup \{j\}$ satisfies $d_{j'} \leq d_j \leq \overline{d}^{(i_0)} \leq \overline{d}^{(i)}$. Thus, it follows that the processing time of the early jobs with due date at most $\overline{d}^{(i)}$ and deadline larger than $\overline{d}^{(i)}$ equals $\sum_{\ell=i+1}^{k} x_\ell$. Because σ is feasible, it follows that $\sum_{\ell=1}^{i} P^{(\ell)} + \sum_{\ell=i+1}^{k} x_\ell \leq \overline{d}^{(i)}$, *i.e.* Item 2 holds.

($\Longrightarrow$:) It remains to show the forward direction. We construct a schedule σ_j with the jobs from S as well as j being early as follows: We start with jobs from S and the jobs with deadline at most $\overline{d}^{(i_0-1)}$ (in the same relative order as they are in σ). Afterwards, we schedule job j, followed by the remaining jobs sorted according to increasing deadline.

First, we show that σ_j is a feasible schedule. Note that for each job from S as well as the jobs with deadline at most $\overline{d}^{(i_0-1)}$, their completion time can only decrease. Consequently, the feasibility of σ implies that these jobs are completed before their deadline. Next, consider a job j' with deadline $\overline{d}^{(i)}$ for some $i \geq i_0$. Before j', all jobs with deadline $\overline{d}^{(\ell)}$ with $\ell < i$, potentially other jobs with deadline $\overline{d}^{(i)}$, and all early jobs from $1, \ldots, j$ are scheduled. Thus, job j' is completed at time $\sum_{\ell=1}^{i} P^{(i)} + \sum_{\ell=i+1}^{k} x_\ell \leq \overline{d}^{(i)}$ where the inequality holds by Item 2. This implies that σ_j is a feasible schedule.

Next, we show that $S \cup \{j\}$ are early. All jobs from S are early as their completion time can only decrease. Job j is completed at time $\sum_{\ell=1}^{i_0-1} P^{(\ell)} + \sum_{\ell=i_0}^{k} x_\ell + p_j \leq d_j$ by Item 1. Therefore, job j is early. ◀

Note that the criterion from Lemma 3.3 is independent from the set of early jobs S, so indeed the dynamic program does not need to store the early jobs. We finally give the dynamic program and show its correctness in the theorem below.

▶ **Theorem 3.4.** *$1|\overline{d}_j|\sum w_j U_j$ can be solved in $O(P^k \cdot k \cdot n)$ time, where k is the number of different deadlines.*

Proof. We give the following dynamic program computing a solution. First, we order the jobs according to ascending due date (we assume that $d_1 \leq d_2 \leq \ldots \leq d_n$). For each $j \in [n]$, the dynamic programming table τ_j contains an entry $\tau_j[x_1, \ldots, x_k]$ for each $(x_1, \ldots, x_k) \in \{0, \ldots, P^{(1)}\} \times \cdots \times \{0, \ldots, P^{(k)}\}$. This entry shall equal $v_j[x_1, \ldots, x_k]$; that is, the maximum number w such that there exists a $(j; x_1, \ldots, x_k)$-compatible schedule σ which schedules a subset $S \subseteq [j]$ of total weight w early; if no $(j; x_1, \ldots, x_k)$-compatible schedule exists, then $\tau_j[x_1, \ldots, x_k] = -\infty$. The maximum of $\tau_n[x_1, \ldots, x_k]$ over all $(x_1, \ldots, x_k) \in \{0, \ldots, P^{(1)}\} \times \cdots \times \{0, \ldots, P^{(k)}\}$ will then yield the value of an optimal schedule.

Initialization. We check whether there is a feasible schedule (this can be done e.g. using Observation 3.2). If so, then we set $\tau_0[0, \ldots, 0] := 0$ while otherwise we set $\tau_0[0, \ldots, 0] := -\infty$. For all $(x_1, \ldots, x_k) \neq (0, \ldots, 0)$, we set $\tau_0[x_1, \ldots, x_k] := -\infty$.

Update. Let $j \in [n]$ and assume that job j has deadline $\overline{d}^{(i)}$. Fix $(x_1, \ldots, x_k) \in \{0, \ldots, P^{(1)}\} \times \cdots \times \{0, \ldots, P^{(k)}\}$. First, we check whether Items 1 and 2 of Lemma 3.3 are satisfied. If yes, then we set

$$\tau_j[x_1, \ldots, x_k] := \max\Big\{\tau_{j-1}[x_1, \ldots, x_k], w_j + \tau_{j-1}[x_1, \ldots, x_{i-1}, x_i - p_j, x_{i+1}, \ldots, x_k]\Big\}.$$

Otherwise, we set $\tau_j[x_1, \ldots, x_k] := \tau_{j-1}[x_1, \ldots, x_k]$.

Optimal Solution. The minimum weighted number of tardy jobs in an optimal schedule is given by taking the minimum of $W - \tau_n[x_1, \ldots, x_k]$ over all $(x_1, \ldots, x_k) \in \{0, \ldots, P^{(1)}\} \times \cdots \times \{0, \ldots, P^{(k)}\}$. An optimal schedule can be found by using backtracking to compute the set of early jobs and then applying Observation 3.2.

Correctness. Clearly, $\tau_0[x_1,\dots,x_k] = \upsilon_0[x_1,\dots,x_k]$ for every $(x_1,\dots,x_k) \in \{0,\dots,P^{(1)}\} \times \cdots \times \{0,\dots,P^{(k)}\}$. Consider $\tau_j[x_1,\dots,x_k]$, where job j has deadline $\overline{d}^{(i)}$. First, we show $\tau_j[x_1,\dots,x_k] \geq \upsilon_j[x_1,\dots,x_k]$. Let σ be a schedule witnessing the value of $\tau_j[x_1,\dots,x_k]$ and S the corresponding set of early jobs from $[j]$. If $j \notin S$, then we have $\upsilon_j[x_1,\dots,x_k] = \upsilon_{j-1}[x_1,\dots,x_k] = \tau_{j-1}[x_1,\dots,x_k] \leq \tau_j[x_1,\dots,x_k]$. Otherwise, we have $\upsilon_j[x_1,\dots,x_k] = w_j + \upsilon_{j-1}[x_1,\dots,x_{i-1},x_i-p_j,x_{i+1},\dots,x_k] = w_j + \tau_{j-1}[x_1,\dots,x_{i-1},x_i-p_j,x_{i+1},\dots,x_k]$ and Lemma 3.3 implies that Items 1 and 2 of Lemma 3.3 are satisfied. Thus, $\tau_j[x_1,\dots,x_k] \geq w_j + \tau_{j-1}[x_1,\dots,x_k] = \upsilon_j[x_1,\dots,x_k]$.

We finish the proof of correctness by showing that $\tau_j[x_1,\dots,x_k] \leq \upsilon_j[x_1,\dots,x_k]$. If $\tau_j[x_1,\dots,x_k] = -\infty$, then there is nothing to show, so assume $\tau_j[x_1,\dots,x_k] > -\infty$. If we have $\tau_j[x_1,\dots,x_k] = \tau_{j-1}[x_1,\dots,x_k]$, then we have $\tau_j[x_1,\dots,x_k] = \tau_{j-1}[x_1,\dots,x_k] = \upsilon_{j-1}[x_1,\dots,x_k] \leq \upsilon_j[x_1,\dots,x_k]$. So assume that we set $\tau_j[x_1,\dots,x_k] = w_j + \tau_{j-1}[x_1,\dots,x_{i-1},x_i-p_j,x_{i+1},\dots,x_k]$. This implies that Items 1 and 2 from Lemma 3.3 are satisfied. Because $\tau_{j-1}[x_1,\dots,x_{i-1},x_i-p_j,x_{i+1},\dots,x_k] = \upsilon_{j-1}[x_1,\dots,x_{i-1},x_i-p_j,x_{i+1},\dots,x_k] \neq -\infty$, Lemma 3.3 implies that we can take the schedule corresponding to $\upsilon_{j-1}[x_1,\dots,x_{i-1},x_i-p_j,x_{i+1},\dots,x_k]$ and additionally schedule job j early. Thus, we have $\upsilon_j[x_1,\dots,x_k] \geq w_j + \upsilon_{j-1}[x_1,\dots,x_{i-1},x_i-p_j,x_{i+1},\dots,x_k] = \tau_j[x_1,\dots,x_k]$.

Running Time. The dynamic programming table contains $P^k \cdot n$ entries, each of which can be computed in $O(k)$ time. The claimed running time follows. ◀

We remark that the ETH-based lower bound from Theorem 3.1 implies that the exponent is optimal up to a factor of $O(\log k)$.

3.3 Few jobs with nontrivial deadlines

Having the intractability results from Theorem 3.1 in mind, we now consider a larger parameter, namely the number of jobs with a nontrivial deadline. Throughout this section, we denote the set of jobs having a nontrivial deadline by $\overline{J}$, and we set $k = |\overline{J}|$. We show that $1|\overline{d}_j|\sum w_jU_j$ can be solved in $O(2^k \cdot P \cdot n)$ time.

The basic idea is that for each job with a deadline, we can guess whether the job is early or tardy. Then, we adapt the due dates of these jobs according to Observation 3.2, resulting in modified due dates d_j^*. Finally, we significantly increase the weight of each job in $\overline{J}$ and then call a known algorithm for $1||\sum w_jU_j$ with respect to the modified due dates d_j^* and weights w_j^*.

▶ **Theorem 3.5.** *$1|\overline{d}_j|\sum w_jU_j$ can be solved in $O(2^k \cdot P \cdot n)$ time, where k is the number of jobs with nontrivial deadline.*

Proof. Let $\overline{J}$ be the set of jobs with a nontrivial deadline. First, we guess the subset of tardy jobs $\overline{U} \subseteq \overline{J}$ with nontrivial deadlines. We then set modified deadlines for all jobs according to Observation 3.2 and using the set $\overline{S} = \overline{J} \setminus \overline{U}$ as the set of early jobs. Thus, we have $d_j^* = d_j$ if $j \in \overline{S}$, and $d_j^* = \overline{d}_j$ if $j \in \overline{U}$. Moreover, we set $w_j^* = W + 1$ for all jobs $j \in \overline{J}$, and $w_j^* = w_j$ for all other jobs $j \in \{1,\dots,n\} \setminus \overline{J}$.

For these modified due dates d_j^* and weights w_j^*, we run the $O(Pn)$-time algorithm for $1||\sum w_jU_j$ [16]. If the algorithm returns a schedule σ where the total weight of tardy jobs is at most W, then we store this schedule as a potential solution for the unmodified instance. Otherwise, we conclude that there is no feasible schedule with respect to our guess of the tardy jobs $\overline{U} \subseteq \overline{J}$. Our algorithm finally outputs the potential solution with the minimum weight of tardy jobs.

By Observation 3.2, the set of feasible schedules where each job from $\overline{J} \setminus \overline{U}$ is early is the set of schedules where each job from $\overline{J}$ is early with respect to the modified due dates. This corresponds to schedules where the total weight of tardy jobs with respect to the modified weights is at most W. Thus, the potential solution corresponding to the current guess $\overline{U} \subseteq \overline{J}$ is a schedule minimizing the total weighted number of tardy jobs under the additional constraint that precisely the $\overline{U}$ jobs of $\overline{J}$ are tardy. It follows that for the guess $\overline{U} = U(\sigma^*)$ for some optimal schedule σ^*, the algorithm returns an optimal schedule. ◀

For the unweighted case (i.e., $w_j = 1$ for each $j \in J$), we can get an FPT-algorithm (even for binary encoded numbers) by following the same approach of guessing which of the jobs with nontrivial deadlines are tardy.

▶ **Theorem 3.6.** *$1|\overline{d}_j|\sum U_j$ can be solved in $O(2^k \cdot n \log n)$ time where k is the number of jobs with a nontrivial deadline.*

Proof. Let $\overline{J}$ be the set of jobs with a nontrivial deadline. First, we guess the subset $\overline{S} \subseteq \overline{J}$ of early jobs with nontrivial deadlines. Observation 3.2 now reduces $1|\overline{d}_j|\sum U_j$ together with the guess $\overline{S}$ to a variation of $1||\sum U_j$ where we are additionally given a set $\overline{J}$ of jobs which have to be early. This problem is known to be solvable in $O(n \log n)$ time [21]. As there are 2^k possible guesses for $\overline{S}$, the running time follows. ◀

4 The $1|\overline{d}_j|\sum w_j C_j$ problem

We next examine the objective of minimizing the total weighted completion time. We remark that the unweighted variant of $1|\overline{d}_j|\sum w_j C_j$, the $1|\overline{d}_j|\sum C_j$ problem, is solvable in $O(n \log n)$ time [22]. Weak NP-hardness of $1|\overline{d}_j|\sum w_j C_j$ was shown by Lenstra *et al.* [17], using only a single job with a nontrivial deadline. Therefore, unless P=NP, there is no XP-algorithm for $1|\overline{d}_j|\sum w_j C_j$ with binary encoding parameterized by the number of jobs with nontrivial deadlines.

Below we strengthen Lenstra's reduction, and show that $1|\overline{d}_j|\sum w_j C_j$ is W[1]-hard when parameterized by the number of jobs with nontrivial deadlines even if all numbers are encoded in unary. Note that this implies also W[1]-hardness for the problem when the number of different deadlines is taken as a parameter. To compliment our hardness result, we present algorithm running in $O(k \cdot n \cdot P^{2k-2})$ time, where k is the number of different deadlines.

4.1 Hardness

We adapt the reduction from Lenstra *et al.* [17] to show strong NP-hardness and W[1]-hardness parameterized by the number of jobs with nontrivial deadline (also for unary encoding). We will reduce from Bin Packing restricted to instances where each bin must be filled exactly.

Exact Bin Packing
Input: t items with sizes $a_1, \ldots, a_t$, a bin size B, and the number K of bins.
Question: Is there an assignment of the items to the bins such that each bin contains items of total size exactly B?

Exact Bin Packing is strongly NP-hard [6] and W[1]-hard parameterized by the number of bins, even if all numbers are encoded in unary [13]. Let $(a_1, \ldots, a_t; B; K)$ be an instance of Exact Bin Packing. We create an instance of $1|\overline{d}_j|\sum w_j C_j$ with $t + K - 1$ jobs as follows:

- For $j \in [t]$, job j has processing time and weight $p_j = a_j = w_j$ and trivial deadline $\overline{d}_j = K \cdot B + K - 1$.
- For $\ell \in [K-1]$, job $t+\ell$ has processing time $p_{t+\ell} = 1$, weight $w_{t+\ell} = 0$, and deadline $\overline{d}_{t+\ell} = \ell \cdot (B+1)$.
- We set the bound on the total weighted completion time to

$$b := \sum_{i_1, i_2 \in [t]} a_{i_1} a_{i_2} + \sum_{\ell=1}^{K-1} (K-\ell) \cdot B.$$

Note that the total processing time of all jobs is $K \cdot B + K - 1 = \overline{d}_j$ for all $j \in [t]$, so only jobs $t+1, \ldots, t+K-1$ have a nontrivial deadline. The idea behind the reduction is as follows: Because $w_j = p_j$ for all $j \in [t]$, the relative order of jobs j_1 and j_2 does not matter: if j_1 is before j_2, then this contributes $w_{j_2} \cdot p_{j_1} = a_{j_1} \cdot a_{j_2}$ to the total completion time (as this increases the completion time of j_2 by p_{j_1}), while otherwise this contributes $w_{j_1} \cdot p_{j_2} = a_{j_1} \cdot a_{j_2}$ to the total processing time. Consequently, the total completion time of a schedule only depends on the completion times of the jobs $t+\ell$ for $\ell \in [K-1]$. As their weight is 0 but they have nonzero processing time, they should be scheduled as late as possible (since they only increase the processing times of other jobs), i.e., directly before their deadline. In any such schedule, the processing time of the jobs between $t+\ell$ and $t+\ell+1$ equals $\overline{d}_{t+\ell+1} - \overline{d}_{t+\ell} - p_{t+\ell+1} = B$, implying a solution to the Exact Bin Packing instance.

▶ **Lemma 4.1.** *The t items can be packed into K bins iff the $t+K+1$ jobs constructed have a feasible schedule with at most b total weighted completion time.*

Proof. ($\Longrightarrow$): Assume that there is a solution to the Exact Bin Packing instance and that the set of items contained in ℓ-th bin is A_ℓ. We denote the set of jobs corresponding to the items from A_ℓ by J_ℓ, *i.e.* $J_\ell := \{j : a_j \in A_\ell\}$. We construct a schedule σ as follows: We start with the jobs from J_1 in arbitrary order, followed by job $t+1$. Afterwards, we schedule the jobs from J_2 followed by $t+2$. We continue scheduling J_ℓ followed by job $t+\ell$ until we schedule job $t+K-1$. Finally, we schedule all jobs from J_K in arbitrary order. This finishes the construction of σ. It remains to show that σ is feasible and has total weighted completion time at most b.

We start with the feasibility of σ. The only jobs with nontrivial deadline are $t+\ell$ for $\ell \in [K-1]$. Job $t+\ell$ is completed after the jobs from $A_1 \cup A_2 \cup \ldots \cup A_\ell$ and jobs $t+1, \ldots, t+\ell$. The processing time of jobs $t+1, \ldots, t+\ell$ is ℓ. Since $\sum_{a \in A_i} a = B$, the processing time of jobs from $A_1 \cup \ldots A_\ell$ is $\ell \cdot B$. Thus, $t+\ell$ is completed at time $\ell + \ell \cdot B = \overline{d}_{t+\ell}$. Consequently, σ is feasible.

We continue by showing that the weighted completion time of σ is at most b. In order to analyze the total completion time, we split it into three parts: First, jobs $t_1, \ldots, t+K-1$ have weight 0 and thus their weighted completion time is 0. Second, we consider the weighted completion time caused by some job j being scheduled after some job $t+\ell$ for $j \in [t]$ and $\ell \in [K-1]$. The jobs scheduled after $t+\ell$ are $J_{\ell+1} \cup J_{\ell+2} \cup \ldots \cup J_K$ as well as $t+\ell+1, \ldots, t+K-1$. Thus, the weight of all jobs scheduled after $t+\ell$ is

$$\sum_{i=\ell+1}^{K} \sum_{j \in J_i} w_j = \sum_{i=\ell+1}^{K} \sum_{a \in A_i} a = \sum_{i=\ell+1}^{K} B = (K-\ell) \cdot B.$$

Finally, we consider the weighted completion time caused by some job j_1 being scheduled before some job j_2 for $j_1, j_2 \in [t]$. For each $j_1, j_2 \in [t]$, job j_1 is either scheduled before j_2,

job j_2 is scheduled before j_1, or $j_1 = j_2$. In all cases, this contributes $a_{j_1} \cdot a_{j_2} = w_{j_1} \cdot p_{j_2} = w_{j_2} \cdot p_{j_1}$ to the total weighted completion time. Summing all three parts together, the total weighted completion time is

$$\sum_{\ell=1}^{K-1} (K-\ell) \cdot B + \sum_{j_1,j_2 \in [t]} a_{j_1} a_{j_2} = b.$$

($\Longleftarrow$): In the converse direction, assume that there is a feasible schedule σ with total weighted completion time at most b. As argued in the forward direction, for each $j_1, j_2 \in [t]$ there is a contribution of $a_{j_1} \cdot a_{j_2}$ to the total weighted completion time. We may assume without loss of generality that σ schedules job $t+\ell$ before job $t+\ell'$ for all $\ell < \ell'$ as jobs $t+\ell$ and $t+\ell'$ have the same weight and processing time, but $\overline{d}_{t+\ell} < \overline{d}_{t+\ell'}$.

Job $t+\ell$ contributes 1 to the completion time of all jobs scheduled after $t+\ell$. Let $J^{>t+\ell}$ be the set of jobs from $[t]$ which are scheduled after $t+\ell$, and let $J^{t+\ell}$ be the jobs from $[t]$ which are scheduled before $t+\ell$. The total processing time of $J^{t+\ell}$ cannot exceed $\ell \cdot B$ as $\overline{d}_{t+\ell} = \ell \cdot (B+1)$, and jobs $t+1, \ldots, t+\ell-1$ are scheduled before $t+\ell$. Consequently, we have

$$\sum_{j \in J^{>t+\ell}} w_j = \sum_{j \in J^{>t+\ell}} a_j = \sum_{j \in [t]} a_j - \sum_{j \in [t] \setminus J^{>t+\ell}} a_j = K \cdot B - \sum_{j \in J^{<t+\ell}} p_j \geq K \cdot B - \ell \cdot B$$

where equality holds if and only if job $t+\ell$ is completed precisely at time $\overline{d}_{t+\ell}$. Because $b = \sum_{i_1,i_2 \in [t]} a_{i_1} a_{i_2} + \sum_{i=\ell}^{K-1} (K-\ell) \cdot B$, this implies that each job $t+\ell$ is completed precisely at time $\overline{d}_{t+\ell}$. Thus, between jobs $t+\ell$ and $t+\ell+1$, jobs of processing time exactly B are scheduled.

Consequently, we construct a solution to the Bin Packing instance as follows: If job j is scheduled after $t+\ell-1$ but before $t+\ell$ for some $\ell \in \{2, 3, \ldots, K-1\}$, then we assign a_j to the ℓ-th bin. If j is scheduled before $t+1$, then we assign a_j to the first bin. If j is scheduled after $t+K-1$, then we assign a_j to the K-th bin. Thus, the t items can all be packed into K bins of size B each. ◀

Recall that Exact Bin Packing is strongly NP-hard, W[1]-hard when parameterized by K, and does not admit an $f(K) \cdot n^{o(K/\log K)}$-time algorithm assuming ETH even if all numbers are encoded in unary [13]. Thus, as the construction above constructs $K-1$ jobs with nontrivial deadlines, we get the following Theorem directly from Lemma 4.1:

▶ **Theorem 4.2.** *Let k denote the number of jobs with nontrivial deadline in a $1|\overline{d}_j|\sum w_j C_j$ instance. Then the $1|\overline{d}_j|\sum w_j C_j$ problem*

1. *is strongly NP-hard,*
2. *is W[1]-hard parameterized by k even when all numbers are encoded in unary, and*
3. *admits no $f(k) \cdot P^{o(k/\log k)}$-time algorithm assuming ETH.*

4.2 Constant number of deadlines

We complement the W[1]-hardness in case of unary encoding shown in the previous subsection by presenting a pseudo-polynomial time algorithm for a constant number of different deadlines. Similar to Section 3.2, the first step of the dynamic program consists of sorting the jobs in a favorable manner. Further, the dynamic program contains a table τ_j for each $j \in [n]$, and these tables contain entries $\tau_j[x_1, \ldots, x_k]$ where x_i encodes the total processing time of the jobs scheduled between $\overline{d}_{i-1}$ and $\overline{d}_i$ (where $\overline{d}_0 := 0$). However, in contrast to Section 3.2,

a schedule corresponding to $\tau_j[x_1, \dots, x_k]$ now only schedules jobs $1, \dots, j$ instead of all jobs, and it may be that the schedule cannot be extended to a feasible schedule for all jobs. Furthermore, the x_i now measure the processing times of jobs scheduled between the $(i-1)$th and ith deadline. Another difference is that we "guess" certain characteristics of an optimal schedule and adapt the instance to these characteristics before starting the dynamic program.

Our dynamic programming processes the jobs from $1, \dots, n$, and computes a table τ_j for each $j \in [n]$. This table has an entry $\tau_j[x_1, \dots, x_k]$ which stores the minimum weighted completion time of a schedule of jobs $\{1, \dots, j\}$ such that

1. the schedule is feasible for $1, \dots, j$, *i.e.* each job from $1, \dots, j$ is completed not after its deadline, and
2. the total processing time of jobs being completed between $\overline{d}^{(i-1)}$ and $\overline{d}^{(i)}$ equals x_i, for every $i \in [k]$ (where we set $\overline{d}^{(0)} := 0$).

We call a schedule fulfilling these two conditions $(j; x_1, \dots, x_k)$-obeying ($(j; x_1, \dots, x_k)$-obeying is the counterpart to $(j; x_1, \dots, x_k)$-compatible from Section 3.2 in the sense that it determines whether a schedule fulfills the conditions for $\tau_j[x_1, \dots, x_k]$).

To compute the above dynamic program, we need to be able to, given an oracle which tells us the set S^i of jobs being completed between $\overline{d}^{(i-1)}$ and $\overline{d}^{(i)}$ for each $i \in [k]$, find an optimal schedule. This can be done easily: As the relative order of the jobs from S^i does not influence the completion times of jobs from S^ℓ for $i \neq \ell$, we can schedule the jobs from S^i independently from the rest. In this subinstance consisting of the jobs from S^i, the deadlines of jobs from S^i are trivial, and so we can use the structure of an optimal schedule for $1||\sum w_j C_j$: An optimal schedule for $1||\sum w_j C_j$ schedules the jobs according to WSPT [22], that is, according to non-decreasing ratio p_j/w_j. Thus, we want to schedule each set S^i according to WSPT. We formalize this argument in the following lemma:

▶ **Lemma 4.3.** *Let σ be an optimal schedule for a $1|\overline{d}_j|\sum w_j C_j$ instance. For $i \in [k]$, let S^i be the set of jobs which are completed after $\overline{d}^{(i-1)}$ but not later than $\overline{d}^{(i)}$ (using $\overline{d}^{(0)} := 0$). Then the jobs from S^i are ordered according to WSPT in σ.*

Proof. Assume towards a contradiction that they are not ordered according to WSPT. Then there is a job $j_1 \in S^i$ directly followed (in σ) by a job $j_2 \in S^i$ with $p_{j_1}/w_{j_1} > p_{j_2}/w_{j_2}$. We claim that the schedule $\sigma_{2,1}$ arising from σ by exchanging jobs j_1 and j_2 is a feasible schedule with smaller total weighted completion time, contradicting the optimality of σ. First, we show the feasibility of $\sigma_{2,1}$. All jobs but j_1 and j_2 are completed at the same time and thus are completed by their deadline by the feasibility of σ. For jobs j_1 and j_2, we have $\min\{\overline{d}_{j_1}, \overline{d}_{j_2}\} \geq \overline{d}^{(i)}$ as σ was a feasible schedule. Further, as j_1 and j_2 are contained in S^i, both jobs are completed not later than $\overline{d}^{(i)}$ in both σ and $\sigma_{2,1}$. Thus, $\sigma_{2,1}$ is feasible. It remains to consider the total weighted completion time of $\sigma_{2,1}$. All jobs except for j_1 and j_2 are completed at the same time and therefore have the same contribution to the weighted completion time. Let t be the starting time of job j_1 in σ. The weighted completion time of j_1 and j_2 in σ is $C := w_{j_1} \cdot (t + p_{j_1}) + w_{j_2}(t + p_{j_1} + p_{j_2})$, while the weighted completion time of j_1 and j_2 in $\sigma_{2,1}$ is $C_{2,1} := w_{j_2} \cdot (t + p_{j_2}) + w_{j_1}(t + p_{j_2} + p_{j_1})$. Thus, we have $C - C_{2,1} = w_{j_2} p_{j_1} - w_{j_1} p_{j_2} > 0$ using $w_{j_1}/p_{j_1} < w_{j_2}/p_{j_2}$ for the inequality. Therefore, $\sigma_{2,1}$ has a smaller total weighted completion time than σ, contradicting the optimality of σ. ◀

Using Lemma 4.3, the basic idea of the dynamic program is the following: We order the jobs according to WSPT, *i.e.* $p_j/w_j \leq p_{j+1}/w_{j+1}$ for all $j \in \{1, \dots, n-1\}$. For each $i \in [k-1]$, we guess when the first job which is completed after $\overline{d}^{(i)}$ starts (there are only P^{k-1} possible guesses). For the sake of simplicity, assume that for each $i \in [k-1]$ the guess

is $\overline{d}^{(i)}$, *i.e.* for each $i \in [k-1]$ there is one job starting at time $\overline{d}^{(i)}$. Recall that the dynamic program $\tau_{j-1}[x_1, \ldots, x_k]$ contains a $(j-1; x_1, \ldots, x_k)$-obeying schedule of $1, \ldots, j-1$ of minimum weighted processing time. There are now up to k possibilities how j is scheduled, namely between $\overline{d}^{(i-1)}$ and $\overline{d}^{(i)}$ for every $i \in [k]$ such that $\overline{d}_j \leq \overline{d}^{(i)}$. Assuming that j is completed between $\overline{d}^{(i-1)}$ and $\overline{d}^{(i)}$, we know how job j is scheduled: As we ordered the jobs according to WSPT, Lemma 4.3 implies that j will be the last job from $1, \ldots, j$ which is completed $\overline{d}^{(i-1)}$ and $\overline{d}^{(i)}$. Thus, it is completed at time $\overline{d}^{(i-1)} + x_i + p_j$. Consequently, this results in a $(j; x_1, \ldots, x_{i-1}, x_i + p_j, x_{i+1}, \ldots, x_k)$-obeying schedule with weighted completion time $\tau_{j-1}[x_1, \ldots, x_k] + w_j \cdot (\overline{d}^{(i-1)} + x_i + w_j)$ if $x_i + p_j \leq \overline{d}^{(i)} - \overline{d}^{(i-1)}$. Therefore, we set $\tau_j[x_1, \ldots, x_k] := \min_{i \in [k]: \overline{d}_j \leq \overline{d}^{(i)}} \tau_{j-1}[x_1, \ldots, x_{i-1}, x_i - p_j, x_{i+1}, \ldots, x_k] + w_j \cdot (\overline{d}^{(i-1)} + x_i)$.

▶ **Theorem 4.4.** *$1|\overline{d}_j| \sum w_j C_j$ can be solved in $O(P^{2k-2} \cdot k \cdot n)$ time, where k is the number of different deadlines.*

Proof. First, we guess for each $i \in [k-1]$ the time t_i when the first job which is completed after $\overline{d}^{(i)}$ starts. Afterwards, we reduce the deadline $\overline{d}^{(i)}$ to t_i. This ensures that there will be no job starting before $\overline{d}^{(i)}$ and being completed after $\overline{d}^{(i)}$.

We order the jobs according to WSPT. The dynamic program contains a table τ_j for each $j \in \{0, \ldots, n\}$. Each such table contains an entry $\tau_j[x_1, \ldots, x_n]$ for $x_i \in \{0, 1, \ldots, \overline{d}^{(i)} - \overline{d}^{(i-1)}\}$ (where $\overline{d}^{(0)} := 0$). This entry contains the minimum weighted completion time of a $(j; x_1, \ldots, x_k)$-obeying schedule. We now formally describe how these values can be computed.

Initialization. We set $\tau_0[0, \ldots, 0] := 0$ and $\tau_0[x_1, \ldots, x_k] := \infty$ otherwise.

Update. Let $j \in [n]$ and assume that job j has deadline $\overline{d}^{(i)}$. Then

$$\tau_j[x_1, \ldots, x_k] := \min_{\ell \in [i]} \left\{ \tau_{j-1}[x_1, \ldots, x_{\ell-1}, x_\ell - p_j, x_{\ell+1}, \ldots, x_k] + w_j \cdot (\overline{d}^{(\ell-1)} + x_\ell) \right\}$$

Optimal Solution. The optimal solution value is $\tau_n[\overline{d}^{(1)}, \overline{d}^{(2)} - \overline{d}^{(1)}, \overline{d}^{(3)} - \overline{d}^{(2)}, \overline{d}^{(4)} - \overline{d}^{(3)}, \ldots, \overline{d}^{(k)} - \overline{d}^{(k-1)}]$. An optimal schedule can be found using backtracking.

Correctness. Clearly, $\tau_0[x_1, \ldots, x_k]$ equals the minimum weighted processing time of a $(0; x_1, \ldots, x_k)$-feasible schedule. Recall that we assume $\overline{d}^{(k)} = P$, so we have $\sum_{i=1}^{k} (\overline{d}^{(i)} - \overline{d}^{(i-1)}) = P$. Because an optimal schedule has no idle time, the total processing time of jobs between $\overline{d}^{(\ell-1)}$ and $\overline{d}^{(\ell)}$ is precisely $\overline{d}^{(\ell)} - \overline{d}^{(\ell-1)}$. Thus, $\tau_n[\overline{d}^{(1)}, \overline{d}^{(2)} - \overline{d}^{(1)}, \overline{d}^{(3)} - \overline{d}^{(2)}, \overline{d}^{(4)} - \overline{d}^{(3)}, \ldots, \overline{d}^{(k)} - \overline{d}^{(k-1)}]$ contains the value of an optimal solution (assuming correctness of the update step).

It remains to show that the update step is correct. Let σ be a $(j; x_1, \ldots, x_k)$-obeying schedule of minimum weighted processing time. Because no job starts before $\overline{d}^{(i)}$ and is completed after $\overline{d}^{(i)}$ as we adapted the deadlines to our initial guess, each job which is completed between $\overline{d}^{(\ell-1)}$ and $\overline{d}^{(\ell)}$ also starts between $\overline{d}^{(\ell-1)}$ and $\overline{d}^{(\ell)}$. Fix $\ell \in [k]$ such that job j is completed in σ between $\overline{d}^{(\ell-1)}$ and $\overline{d}^{(\ell)}$. Note that $\ell \leq i$ as σ is feasible. As we process the jobs in WSPT order, we may assume by Lemma 4.3 that j is the last job from $[j]$ which starts between $\overline{d}^{(\ell-1)}$ and $\overline{d}^{(\ell)}$. Then removing job j results in a schedule

of jobs $1, \dots, j-1$ where the processing time of jobs starting between $\overline{d}^{(\ell'-1)}$ and $\overline{d}^{(\ell')}$ is exactly $x_{\ell'}$ for $\ell' \neq \ell$ and $x_\ell - p_j$ for $\ell' = \ell$. Scheduling j after all other jobs starting between $\overline{d}^{(\ell-1)}$ and $\overline{d}^{(\ell)}$ then results in a completion time of $\overline{d}^{(\ell)} + x_\ell$ for job j. The total weighted completion time of jobs $1, \dots, j-1$ is at least $\tau_{j-1}[x_1, \dots, x_{\ell-1}, x_\ell - p_j, x_{\ell+1}, \dots, x_k]$. Consequently, a $(j; x_1, \dots, x_k)$-obeying schedule has weighted completion time at least $w_j(\overline{d}^{(\ell)} + x_\ell) + \tau_{j-1}[x_1, \dots, x_{\ell-1}, x_\ell - p_j, x_{\ell+1}, \dots, x_k] \geq \tau_j[x_1, \dots, x_k]$.

It remains to show that an $(j; x_1, \dots, x_k)$-obeying schedule has weighted completion time at most $\tau_j[x_1, \dots, x_k]$. For each $\ell \leq i$, combining the schedule from $\tau_j[x_1, \dots, x_{\ell-1}, x_\ell - p_j, x_{\ell+1}, x_k]$ with job j scheduled at time $\overline{d}^{(\ell-1)} + x_\ell$ results in a schedule σ_ℓ with total weighted completion time $\tau_j[x_1, \dots, x_{\ell-1}, x_\ell - p_j, x_{\ell+1}, x_k] + w_j \cdot (\overline{d}^{(\ell-1)} + x_\ell)$. Schedule σ_ℓ completes j before its deadline as $\ell \leq i$ and $x_\ell \leq \overline{d}^{(\ell)} - \overline{d}^{(\ell-1)}$. Thus, σ_ℓ is $(j; x_1, \dots, x_k)$-obeying. Therefore, there is a $(j; x_1, \dots, x_k)$-obeying schedule with weighted completion time at most $\tau_j[x_1, \dots, x_k]$.

Running Time. There are $O(P^{k-1})$ many different guesses (for each $i \in [k-1]$, we guess one time t_i). For each guess, the dynamic programming table as described above contains $O(n \cdot P^k)$ many entries, each of which can be computed in $O(k)$ time. However, note that we only need to consider entries $\tau_j[x_1, \dots, x_k]$ with $\sum_{\ell=1}^{k} x_\ell = \sum_{j'=1}^{j} p_{j'}$. Thus, for each combination of j and $x_1, \dots, x_{k-1}$, there is only one value of x_k which we need to consider. This implies that it suffices to compute $O(n \cdot P^{k-1})$ many entries of the dynamic programming table. Thus, the total running time is $O(k \cdot n \cdot P^{2k-2})$. ◀

We remark that the ETH-based lower bound from Theorem 4.2 implies that the exponent is optimal up to a factor of $O(\log k)$.

5 Conclusion

We initiated the study of the parameterized complexity of scheduling problems with deadlines. While we arrived at a complete FPT-vs.-W[1]-hardness-vs.-XP-classification of $1|\overline{d}_j|\sum w_j U_j$ and $1|\overline{d}_j|\sum w_j C_j$ with respect to the number of different deadlines and the number of jobs with nontrivial deadline, there is still ample room for future work: For example, one might study other parameterizations not focusing on the deadlines such as the number of different processing times. Another direction would be to extend our study to further problems such as $1|\overline{d}_j|\sum T_j$. For this problem, one likely gets similar in Theorem 3.1 by modifying the reduction from [3] similar to Theorem 3.1, but it is unclear on whether a pseudopolynomial-time algorithm for a constant number of different deadlines exists. Lastly, we left open the approximability of both $1|\overline{d}_j|\sum w_j C_j$ and $1|\overline{d}_j|\sum w_j U_j$ (note that $1||\sum w_j U_j$ admits an FPTAS [20]). Due to the strong NP-hardness of $1|\overline{d}_j|\sum w_j C_j$ and $1|\overline{d}_j|\sum w_j U_j$, both problems do not admit an FPTAS unless P=NP. However, the existence of a PTAS or a constant-factor approximation algorithm is open.

References

1 Philippe Baptiste, Federico Della Croce, Andrea Grosso, and Vincent T'kindt. Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances. *Journal of Scheduling*, 13(1):39–47, 2010.

2 Peter Brucker. *Scheduling algorithms (4. ed.)*. Springer, 2004.

3 Rubing Chen and Jinjiang Yuan. Unary NP-hardness of single-machine scheduling to minimize the total tardiness with deadlines. *Journal of Scheduling*, 22(5):595–601, 2019.

4 Rubing Chen and Jinjiang Yuan. Unary NP-hardness of preemptive scheduling to minimize total completion time with release times and deadlines. *Discrete Applied Mathematics*, 304:45–54, 2021.

5 Rubing Chen, Jinjiang Yuan, C.T. Ng, and T.C.E. Cheng. Single-machine scheduling with deadlines to minimize the total weighted late work. *Naval Research Logistics*, 66(7):582–595, 2019.

6 Michael R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.*, 4(4):397–411, 1975.

7 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

8 Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

9 A. M. A. Hariri and Chris N. Potts. Single machine scheduling with deadlines to minimize the weighted number of tardy jobs. *Management Science*, 40(12):1712–1719, 1994.

10 Cheng He, Hao Lin, Yixun Lin, and Junmei Dou. Minimizing total completion time for preemptive scheduling with release dates and deadline constraints. *Foundations of Computing and Decision Sciences*, 39(1):17–26, 2014.

11 Cheng He, Yixun Lin, and Jinjiang Yuan. A note on the single machine scheduling to minimize the number of tardy jobs with deadlines. *European Journal of Operational Research*, 201(3):966–970, 2010.

12 Yumei Huo, Joseph Y.-T. Leung, and Hairong Zhao. Bi-criteria scheduling problems: Number of tardy jobs and maximum weighted tardiness. *European Journal of Operational Research*, 177(1):116–134, 2007.

13 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013.

14 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center*, The IBM Research Symposia Series, pages 85–103. Plenum Press, 1972.

15 Eugene L. Lawler. Scheduling a single machine to minimize the number of late jobs. Technical Report UCB/CSD-83-139, EECS Department, University of California, Berkeley, 1983.

16 Eugene L. Lawler and J. Michael Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969.

17 Jan K. Lenstra, Alexander H.G. Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977.

18 J. Michael Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968.

19 Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems (5. ed.)*. Springer, 2016.

20 Sartaj Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23(1):116–127, 1976.

21 Jeffrey B. Sidney. An extension of Moore's due date algotithm. In *Symposium on the Theory of Scheduling and Its Applications*, pages 393–398. Springer Berlin Heidelberg, 1973.

22 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

23 Long Wan, Jinjiang Yuan, and Zhichao Geng. A note on the preemptive scheduling to minimize total completion time with release time and deadline constraints. *Journal of Scheduling*, 18(3):315–323, 2015.

24 Jinjiang Yuan. Unary NP-hardness of minimizing the number of tardy jobs with deadlines. *Journal of Scheduling*, 20(2):211–218, 2017.