Journal of Scheduling (2024) 27:545–556
https://doi.org/10.1007/s10951-024-00818-9

Serial batching to minimize the weighted number of tardy jobs

Danny Hermelin[1] · **Matthias Mnich**[2] · **Simon Omlor**[3]

Accepted: 25 July 2024 / Published online: 22 October 2024

Abstract
The $1|\text{s-batch}(\infty), r_j|\sum w_jU_j$ scheduling problem takes as input a batch setup time Δ and a set of n jobs, each having a processing time, a release date, a weight, and a due date; the task is to find a sequence of batches that minimizes the weighted number of tardy jobs. This problem was introduced by Hochbaum and Landy (Oper Res Lett 16(2):79–86, 1994); as a wide generalization of KNAPSACK, it is NP-hard. In this work, we provide a multivariate complexity analysis of the $1|\text{s-batch}(\infty), r_j|\sum w_jU_j$ problem with respect to several natural parameters. That is, we establish a classification into fixed-parameter tractable and W[1]-hard problems, for parameter combinations of (i) $\#p$ = number of distinct processing times, (ii) $\#w$ = number of distinct weights, (iii) $\#d$ = number of distinct due dates, (iv) $\#r$ = number of distinct release dates. Thereby, we significantly extend the work of Hermelin et al. (Ann Oper Res 298:271–287, 2018) who analyzed the parameterized complexity of the non-batch variant of this problem without release dates. As one of our key results, we prove that $1|\text{s-batch}(\infty), r_j|\sum w_jU_j$ is W[1]-hard parameterized by the number of distinct processing times and distinct due dates. To the best of our knowledge, these are the first parameterized intractability results for scheduling problems with few distinct processing times. Further, we show that $1|\text{s-batch}(\infty), r_j|\sum w_jU_j$ is fixed-parameter tractable parameterized by $\#d+\#p+\#r$, and parameterized by $\#d+\#w$ if there is just a single release date. Both results hold even if the number of jobs per batch is limited by some integer b.

Keywords Scheduling · Single machine scheduling · Batch scheduling · Weighted number of tardy jobs · Fixed-parameter tractability

1 Introduction

This paper is concerned with the problem of minimizing the total weight of tardy jobs in a non-preemptive single machine (serial) batch scheduling environment. Before describing our results, we first briefly overview the classical non-batch variant of this problem, denoted as $1||\sum w_jU_j$ in Graham's classical three-field notation (Graham et al., 1979). Following this, we describe the extension of $1||\sum w_jU_j$ to the batch scheduling environment and discuss how our results fit into the known state of the art.

M. Mnich and S. Omlor supported by DFG Grant MN 59/4-1.

✉ Matthias Mnich
matthias.mnich@tuhh.de

Danny Hermelin
hermelin@bgu.ac.il

Simon Omlor
simon.omlor@tu-dortmund.de

[1] Ben-Gurion University of the Negev, Beersheba, Israel

[2] Hamburg University of Technology, Institute for Algorithms and Complexity, Hamburg, Germany

[3] Department of Statistics, TU Dortmund University, Dortmund, Germany

1.1 Total weight of tardy jobs on a single machine

One of the most fundamental and prominent scheduling criteria on a single machine is that of minimizing the total weight of tardy jobs in a schedule. Let J be a set of jobs, where each job $j \in J$ has a *processing time* $p_j \in \mathbb{N}$, a *weight* $w_j \in \mathbb{N}$, and a *due date* $d_j \in \mathbb{N}$. We are given a single machine on which to process all the jobs in J. A schedule for this machine corresponds to assigning a *starting time* S_j to each job $j \in J$, so that $S_i \notin [S_j, S_j + p_j)$ for any job $i \neq j$. The term $S_j + p_j$, also denoted C_j, is called the *completion time* of job j. A job $j \in J$ is *tardy* if its completion time exceeds its due date, i.e., if $C_j > d_j$; otherwise, it is *early*. The goal is to find a schedule which minimizes the total weight of all tardy jobs; formally, this means to minimize $\sum_{j\in J} w_jU_j$ where U_j is

a binary indicator variable which takes value 1 if job j is tardy and value 0 otherwise. In Graham's 3-field notation, this problem is denoted as $1||\sum w_jU_j$.

Karp (1972) proved that this problem is (weakly) NP-hard even when all jobs have a common due date (i.e., the $1|d_j = d|\sum w_jU_j$ problem), and in fact this variant is equivalent to the 0/1 KNAPSACK problem. The variant where in addition to a single due date, the weight of each job is equal to its processing time (that is, the problem $1|d_j = d, p_j = w_j|\sum w_jU_j$) is known to be equivalent to the SUBSET SUM problem.

Lawler and Moore (1969) provided a pseudo-polynomial time algorithm for $1||\sum w_jU_j$, whereas Sahni (1976) showed that the problem admits an FPTAS. The variant where all jobs have unit weight (and a single release date), known as the $1||\sum U_j$ problem, is solvable in $O(n \log n)$ time due to an algorithm by Moore (1968). There is also a classical variant where each job $j \in J$ also has a release date $r_j \in \mathbb{N}$, and $S_j \geq r_j$ is required of any schedule. This variant is known to be NP-hard even if jobs have unit weight and there are only two distinct due dates and only two distinct release dates (Karp, 1972).

We approach the serial batching problems with tools from parameterized complexity. There, a computational problem Π
takes as input pairs (x, k) where $x \in \Sigma^\star$ is a string and $k \in \mathbb{N}$ is called a *parameter*. The parameter provides a secondary measurement of the structures inherent to the instance which drive a problem's complexity. The major goal is to show a problem to be "fixed-parameter tractable" for parameter k: We say that problem Π is *fixed-parameter tractable* (or lies in FPT) for parameter k if it admits an algorithm which solves all its instances of size n in time $f(k) \cdot n^{O(1)}$ for some computable function f; such an algorithm is called a *fixed-parameter algorithm*. A weaker condition is when problem Π lies in XP: this means that its instances of size n can be solved by an XP-*algorithm*, which is only required to run in time $n^{f(k)}$. To show that a problem Π is unlikely to be fixed-parameter tractable for a certain parameter k, one shows it to be W[1]-hard by a reduction from a known W[1]-hard problem Π', where the reduction maps instances $(x', k') \in \Pi'$ to instances $(x, k) \in \Pi$ with k depending on k' only. The fastest known algorithms for W[1]-hard problem take time $n^{f(k)}$, which is considered to be inefficient for large instances, even if k is small. For background on parameterized complexity, we refer to the book by Cygan et al. (2015).

Studying the parameterized complexity of scheduling problems is a field of growing interest. Several works design fixed-parameter algorithms for scheduling problems (Hermelin et al., 2019; Knop & Koutecký 2018; Knop et al., 2020; Mnich & Wiese, 2015; van Bevern et al., 2015, 2016, 2017). For background, we refer to the recent survey on the topic (Mnich & van Bevern, 2018).

Most relevant to this paper is a recent result by Hermelin et al. (2018) who studied the parameterized complexity of $1||\sum w_jU_j$. There, the following three parameters are considered for the problem:

- the number $\#d$ of distinct due dates,
- the number $\#p$ of distinct processing times,
- and the number $\#w$ of distinct weights.

Their main results are given in the proposition below:

Proposition 1 (Hermelin et al., 2018) *Problem* $1||\sum w_jU_j$ *can be solved in*

- *time* $f(\#d + \#p) \cdot n^{O(1)}$, *time* $f(\#d + \#w) \cdot n^{O(1)}$, *and in time* $f(\#p + \#w) \cdot n^{O(1)}$ *for some computable function* $f : \mathbb{R} \to \mathbb{R}$.
- *time* $n^{O(\#p)}$, *and in time* $n^{O(\#w)}$.

A special case of this result was already obtained by Etscheid et al. (2017) who presented an $f(\#p) \cdot n^{O(1)}$-time algorithm for the single due date $1|d_j = d|\sum w_jU_j$ problem.

1.2 Batch scheduling

Batch scheduling has received a considerable amount of attention in the scheduling community (Potts & Kovalyov, 2000). The motivation for this line of research stems from the fact that in manufacturing systems items flow between facilities in boxes, pallets, or carts. A set of items assigned to the same container is considered as a *batch*. All items in the same batch leave the facility together, and thus have equal completion time. There are two main batch models that are considered in the literature: In the *parallel* model, the completion time of a batch equals the maximum completion time of any job within it; this model is extensively surveyed by Fowler and Mönch (2022). In the *serial* model, the completion time of a batch equals the starting time plus the setup time plus the sum of processing times of jobs in the batch. The focus of this paper is exclusively on the latter serial model, and thus we will use the terms batch scheduling and serial batch scheduling synonymously. We refer to Potts and Kovalyov (2000) and Webster and Baker (1995) for further reading on the topic, as well as the classical textbooks of Brucker (2007) and Pinedo (2016).

Hochbaum and Landy (1994) studied the generalization of the $1||\sum w_jU_j$ problem to the (serial) batch setting. In this problem, which we denote by $1|\text{s-batch}(\infty)|\sum w_jU_j$, a schedule consists of a partition of the job set J into batches, and a starting time S_B for each batch B such that $S_{B'} \notin [S_B, C_B = S_B + \Delta + \sum_{j \in B} p_j)$ for any batch $B' \neq B$, where Δ is a given *setup time* associated with starting any batch. The completion time of any job $j \in B$ is $C_j = C_B$,

and the starting time is $S_j = S_B$, meaning that all the jobs together in a batch are started and completed at the same time. The goal is again to minimize the total weight of tardy jobs $\sum w_j U_j$. Note that the order of the jobs within each batch is irrelevant and that when $\Delta = 0$ this problem becomes the classical $1||\sum w_j U_j$ problem. Our notation "s-batch(∞)" refers to that there is no bound on the size of each batch.

Hochbaum and Landy (1994) observed that the $1|$s-batch $(\infty)|\sum w_j U_j$ problem is weakly NP-hard (being a direct generalization of $1||\sum w_j U_j$), and provided pseudopolynomial-time algorithms for the problem that are linear in the total sum of job processing times (plus $n \cdot \Delta$), or the maximum due date. Brucker and Kovalyov provided an analogous algorithm which is linear in the total sum of job weights (Brucker & Kovalyov, 1996). Nevertheless, in this paper we are interested in the case where job weights, processing times, or due dates can be arbitrarily large, but the number of different values of each of these parameters (namely, $\#w$, $\#p$, or $\#d$) is relatively small. In this context, the following result of Hochbaum and Landy is very relevant.

Proposition 2 (Hochbaum & Landy, 1994) *Both* $1|$s-batch $(\infty)|\sum U_j$ *and* $1|$s-batch$(\infty), p_j = p|\sum w_j U_j$ *are polynomial-time solvable.*

Baptiste (2000) generalized the result of Hochbaum and Landy (1994) to a polynomial-time algorithm for $1|$s-batch $(\infty), p_j = p, r_j|\sum w_j U_j$, where jobs have equal processing times but individual release dates.

One can also consider the model of bounded batch sizes. Cheng and Kovalyov (2001) argued about the importance of batch sizes bounded by $|B| \leq b$ in real-life applications. Note that for $b = n$ we have the unbounded $1|$s-batch$(\infty)|\sum w_j U_j$ problem, whereas for $b = 1$ one obtains the classical non-batch model $1||\sum w_j U_j$. Cheng and Kovalyov showed that $1|$s-batch$(b)|\sum U_j$ is in XP when parameterized by either $\#p$ or $\#d$, which means that for *constant* parameter values the problem can be solved in polynomial time:

Proposition 3 (Cheng & Kovalyov, 2001) *Problem* $1|s\text{-}batch(b)|\sum U_j$ *can be solved in time* $n^{O(\#p)}$, *and in time* $n^{O(\#d)}$.

1.3 Our contributions

We provide a multivariate complexity analysis of $1|$s-batch $(\infty)|\sum w_j U_j$ and related variants: Problem $1|$s-batch(∞), $r_j|\sum w_j U_j$ where jobs also have release dates, problem $1|$s-batch$(b)|\sum w_j U_j$ where there is a bound b on the batch size.

The standard batch model In the first part of the paper, we study the $1|$s-batch$(\infty)|\sum w_j U_j$ problem without release dates or batch size restrictions. We show that almost all results of Proposition 1 regarding the $1||\sum w_j U_j$ problem extend to the batch setting.

Theorem 1 *Problem* $1|$s-batch$(\infty)|\sum w_j U_j$ *can be solved in time* $n^{O(\#p)} \cdot O(\#d)$, *in time* $n^{O(\#w)} \cdot O(\#d)$, *in time* $f(\#d + \#p) \cdot n^{O(1)}$, *and in time* $f(\#d + \#w) \cdot n^{O(1)}$.

The XP-algorithms alluded to in this theorem are based on dynamic programming, while the fixed-parameter algorithms are based on an elegant reduction to the non-batch case. Note that the first and second item of the theorem generalize the results by Hochbaum and Landy stated in Proposition 2.

Release dates We first extend the result of Baptiste (2000) to jobs with arbitrary processing times, by showing that $1|$s-batch$(\infty), r_j|\sum w_j U_j$ can be solved in time $n^{f(\#d+\#p+\#w)}$ and in time $n^{f(\#p+\#r+\#w)}$. Next, we show that these run times are tight, in the sense that the dependence on the parameters can very likely not be removed from the degree of n. Specifically, we prove that $1|$s-batch$(\infty), r_j|\sum w_j U_j$ is W[1]-hard for either parameter $\#d + \#p$ or $\#p + \#r$, and therefore is not fixed-parameter tractable under the standard hypothesis that FPT $\neq$ W[1].

Theorem 2 *Problem* $1|s\text{-}batch(\infty), r_j|\sum w_j U_j$ *is* W[1]*-hard when parameterized by* $\#d + \#p$, *and is* W[1]*-hard when parameterized by* $\#p + \#r$. *Furthermore, the problem is solvable in time* $n^{f(\#d+\#p+\#w)}$, *and in time* $n^{f(\#p+\#r+\#w)}$.

To the best of our knowledge, Theorem 2 is the first W[1]-hardness result for *any* scheduling problem parameterized by the number of distinct processing times $\#p$. In particular, whether or not $P||C_{\max}$ (makespan minimization on an unbounded number of parallel machines) is W[1]-hard for this parameter is a well-known open problem (Goemans & Rothvoss, 2020; Mnich & van Bevern, 2018), and this question is also open for $1||\sum w_j U_j$ (Hermelin et al., 2018).

Batch restrictions In the final part of the paper, we show that the $f(\#d + \#w) \cdot n^{O(1)}$-time algorithm in the second part of Proposition 1 can be generalized to the setting where each batch contains at most b jobs; this setting was proposed by Cheng and Kovalyov (2001).

Theorem 3 *The following problems are fixed-parameter tractable:*

- $1|s\text{-}batch(b)|\sum w_j U_j$ *for parameter* $\#d + \#w$.
- $1|s\text{-}batch(b), r_j|\sum w_j U_j$ *for parameter* $\#d + \#p + \#r$.

In particular, this improves the result of Cheng and Kovalyov (2001), as our algorithm runs in time $f(\#d) \cdot n^{O(1)}$ for the unweighted version $1||$s-batch$(\infty)| \leq b|\sum U_j$.

A summary of our results is given in Table 1.

As this moment, we do not know what the parameterized complexity of the problem for parameter $\#p + \#w$ could be; we thus leave this question for future research. Note that our results describing that taking either release dates, or due dates, as a parameter leads to a similar levels of parameterized complexity of the studied problems, which reflects

Table 1 Summary of results

Problem variant	Parameters	Result	Reference
$1\|\text{s-batch}(\infty)\|\sum w_jU_j$	$\#p$	XP	Theorem 1
	$\#w$	XP	Theorem 1
	$\#d+\#p$	FPT	Theorem 1
	$\#d+\#w$	FPT	Theorem 1
$1\|\text{s-batch}(\infty), r_j\|\sum w_jU_j$	$\#p+\#r$	W[1]-hard	Theorem 2
	$\#d+\#p$	W[1]-hard	Theorem 2
	$\#d+\#p+\#w$	XP	Theorem 2
	$\#p+\#r+\#w$	XP	Theorem 2
$1\|\text{s-batch}(b)\|\sum w_jU_j$	$\#d+\#w$	FPT	Theorem 3
$1\|\text{s-batch}(b), r_j\|\sum U_j$	$\#d+\#p+\#r$	FPT	Theorem 3

the intuitive notion of time reversibility (which refers to the intuition that reversing an optimal schedule for an instance leads to an optimal schedule for an instance with the roles of release dates and due dates interchanged).

2 Preliminaries

We are given a set J of n jobs. Each job $j \in J$ has an integral processing time p_j, a weight w_j, a due date d_j and a release dates r_j. Our task is to find a schedule that assigns to each job a starting time $S_j \geq r_j$, such that for any two jobs j, j' we have $S_{j'} \notin [S_j, S_j + p_j)$ (i.e., only one job is processed at each time slot). Jobs that complete in time (before their due date) are called *early*; the other jobs (for which $S_j + p_j > d_j$) are *tardy*.

Our focus in this paper is to schedule jobs in batches. Formally, the problem $1|\text{s-batch}(\infty), r_j|\sum w_jU_j$ takes as input an integer batch setup time Δ and a set J of n jobs each of which is characterized by an integer due date d_j, integer processing time p_j, and integer release date r_j and a positive integer weight w_j. Our task is find a *schedule* for J, which is a partition the job set J into a set $\mathcal{B}$ of *batches*, and then assign a starting time S_B to each batch $B \in \mathcal{B}$ such that no two batches are processed at the same time. The processing time of a batch B is given by $p_B = \Delta + \sum_{j \in J_B} p_j$. The completion time C_j of each job $j \in J_B$ is determined by the completion time of the last job in B, that is, $C_j = S_B + p_B$. The objective is to find a schedule which minimizes the weighted number $\sum_{j \in J} w_jU_j$ of tardy jobs, where $U_j = 1$ if $C_j > d_j$ and $U_j = 0$ otherwise.

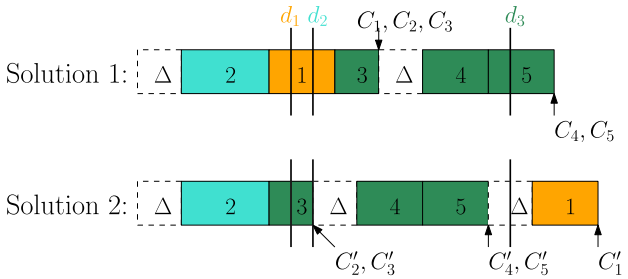

Fig. 1 An example for batch scheduling with 5 jobs. In Solution 1, jobs 1 and 5 are tardy. In Solution 2, only job 1 is tardy. Tardy jobs can be moved to the end of the schedule without increasing the weight of the tardy jobs

3 The standard batch model

In this section, we present algorithms for the basic $1|\text{s-batch}(\infty)|\sum w_jU_j$ problem, providing a complete proof for Theorem 1. Note that in the setting where all jobs are released at the same time and the batch sizes are not restricted, we can schedule the early jobs in order of the due dates. We use this helpful observation by Hochbaum and Landy (1994) multiple times: We illustrate it by an example in Fig. 1.

Lemma 1 (Hochbaum & Landy, 1994) *Any instance of $1|\text{s-batch}(\infty)|\sum w_jU_j$ admits an optimal solution in which all early jobs are in earliest due date (EDD) order. That is, for any two jobs j, j' scheduled in different batches $B^{(-1)}(j)$ and $B^{(-1)}(j')$ with $S_{B^{(-1)}(j)} < S_{B^{(-1)}(j')}$, we have $d_j < d_{j'}$.*

Our first result uses this observation to yield an elegant reduction to the non-batch case. We use following notation to order the due dates: $d^{(1)} < d^{(2)} < \cdots < d^{(\#d)}$. Further, we set $d^{(0)}$ to be the smallest release date.

3.1 Fixed-parameter algorithms

We begin by presenting fixed-parameter algorithms for $1|\text{s-batch}(\infty)|\sum w_jU_j$ for parameter $\#d + \#p$, and for $\#d + \#w$.

Lemma 2 *Problem* $1|s\text{-}batch(\infty)|\sum w_jU_j$ *is solvable in time* $f(\#d+\#p)\cdot n^{O(1)}$, *and in time* $f(\#d+\#w)\cdot n^{O(1)}$.

Proof Let J denote the job set of our $1|\text{s-batch}(\infty)|\sum w_jU_j$ instance. We first observe that there is an optimal schedule in which at most one batch completes within each interval $(d^{(i-1)}, d^{(i)}]$, for each $i \in \{1,\ldots,\#d\}$; if two or more batches end in $(d^{(i-1)}, d^{(i)}]$, then these batches can be combined into a single batch without creating new tardy jobs. The second observation is that if no batch ends in $(d^{(i-1)}, d^{(i)}]$, then all jobs with due date $d^{(i)}$ that are completed early must be in batches ending at $d^{(i-1)}$ or earlier. We next use these observations to reduce our $1|\text{s-batch}(\infty)|\sum w_jU_j$ instance J into $2^{O(\#d)}$ instances of the non-batch $1||\sum w_jU_j$ problem, each with the same number of processing times, weights, and due dates as in J. Combined with the fixed-parameter algorithms for $1||\sum w_jU_j$ given by Hermelin et al. (2018), this will provide a proof for the theorem.

For each $i \in \{1,\ldots,\#d\}$, we guess whether some batch ends in $(d^{(i-1)}, d^{(i)}]$ in an optimal solution. Let $I \subseteq \{1,\ldots,\#d\}$ be the set of indexes i such that some batch ends in $(d^{(i-1)}, d^{(i)}]$ with respect to our guess. For an index $\ell \in \{1,\ldots,\#d\}$, let $I_{\leq\ell} = \{i \in I \mid i \leq \ell\}$ be the set of indices in I smaller or equal to ℓ, and let $i(\ell) = \max\{i \in I \mid i \leq \ell\}$ be the largest index in I that is less than or equal to ℓ. We construct an instance J_I of $1||\sum w_jU_j$ corresponding to I by replacing the due date $d_j = d^{(\ell)}$ of each job $j \in J$ with an alternative due date $d'_j = d^{(i(\ell))} - |I_{\leq\ell}|\cdot\Delta$; all other job parameters remain the same in J_I.

Consider some set of indices $I \subseteq \{1,\ldots,\#d\}$, and let J_I be the corresponding $1||\sum w_jU_j$ instance. We can convert a schedule of J_I as follows: We note that

$$\sum_{j \text{ is early and } d'_j \leq d} p_j \leq d$$

for all due dates d. We construct $|I|+1$ batches. The first $|I|$ batches are denoted by B_i for $i \in I$ and are processed in increasing order, i.e. if $i < i'$ then B_i is processed before $B_{i'}$. Let j be an early job (i.e. j' is early) with $d_j = d^{(\ell)}$ for some ℓ. Then we assign j to batch $B_{i(\ell)}$. We conclude that j will be early, as the completion time of $B_{i(\ell)}$ is equal to

$$\begin{aligned} C_{B_{i(\ell)}} &= |I_{\leq\ell}|\Delta + \sum_{\substack{j' \text{ is early} \\ \text{and } d_{j'} \leq d_j}} p_{j'} \leq |I_{\leq\ell}|\Delta + d'_j \\ &= d^{i(\ell)} \leq d_j. \end{aligned}$$

Conversely, consider any schedule for J that schedules at most one batch ending in each interval of consecutive due dates, and let $I \subseteq \{1,\ldots,\#d\}$ be the corresponding set of indices. Then any early job $j \in J$ with $d_j = d^{(\ell)}$ has $C_j \leq d^{i(\ell)}$, and so its completion time in the non-batch setting under the same ordering of early jobs is at most

$$C_j - |I(\leq\ell)|\Delta \leq d^{i(\ell)} - |I(\leq\ell)|\Delta = d'_j.$$

It follows that an optimal schedule for our original $1|\text{s-batch}(\infty)|\sum w_jU_j$ instance corresponds to the schedule with the minimum weight of tardy jobs among all optimal schedules for instances $J_I, I \subseteq \{1,\ldots,\#d\}$. The lemma then follows, since there are $2^{\#d}$ instances J_I, each of which can be solved in $f(\#d+\#p)\cdot n^{O(1)}$ or $f(\#d+\#w)\cdot n^{O(1)}$ time using the algorithm by Hermelin et al. (2018). □

3.2 XP-algorithms

Next, we consider the parameterizations by only $\#p$. Assume that our input job set $\{1,\ldots,n\}$ is ordered such that $d_1 \leq \cdots \leq d_n$ (i.e. ordered according to EDD). Due to Lemma 1, there is an optimal schedule where any job $j \in J$ is either late, or it is scheduled after the early jobs in $\{1,\ldots,j-1\}$. Thus, an optimal schedule for jobs $\{1,\ldots,j\}$ can be found by appending j to some schedule of jobs $\{1,\ldots,j-1\}$.

Lemma 3 *Problem* $1|s\text{-}batch(\infty)|\sum w_jU_j$ *is solvable in time* $n^{O(\#p)}\cdot O(\#d)$.

Proof Let $J = \{1,\ldots,n\}$ denote our job set ordered according to EDD, and let $p^{(1)} < \cdots < p^{(\#p)}$ denote the different processing times of all jobs in J. For increasing values of $j \in \{1,\ldots,n\}$, we compute a table W_j which has $n^{O(\#p)}$ entries and corresponds to jobs in $\{1,\ldots,j\}$.

The table W_j will be indexed by a $\#p$-dimensional vector $I \in \{1,\ldots,n\}^{\#p}$, and integer $b \in \{0,1,\ldots,n\}$, and a due date $d \in \{d_1,\ldots,d_n\}$. The invariant that our algorithm will maintain is that $W_j[I,b,d]$ equals the minimum total weight of tardy jobs in a schedule for jobs $\{1,\ldots,j\}$ with the following properties:

1. The early jobs are scheduled in EDD fashion as in Lemma 1.
2. There are exactly b batches containing exactly $I[i]$ early jobs, $i \in \{1,\ldots,\#p\}$, with processing time $p^{(i)}$, scheduled consecutively starting from time 0.
3. The earliest due date among all jobs in the last batch is at least d.

Note that there exists a vector I and integers b and d such that the optimal schedule for J satisfies all properties required from a schedule corresponding to entry $W_n[I,b,d]$ and all jobs in the first b batches are early.

In the beginning, we set $W_j[I,b,d] = \sum_{i=1}^{j} w_i$ if $I = \emptyset$ and $W_j[I,b,d] = \infty$ otherwise. Fix $j \in \{1,\ldots,n\}$, and consider an entry $W_j[I,b,d]$ of W_j. Let $p^{(\ell)} = p_j$ be the processing time of j for $\ell \in \{1,\ldots,\#p\}$. Let I_ℓ be the vector which coincides with I on every coordinate, except for the ℓ^{th} coordinate for which it is equal to $I[\ell]-1$. If the ℓ^{th} coordinate of I is 0, then j is tardy by the definition and

we set $W_j[I,b,d] = W_{j-1}[I,b,d] + w_j$. If $\sum_{i=1}^{\#p} I[i] \cdot p^{(i)} + b\Delta > d$, then job j will be late if it is among the jobs scheduled in the first b batches. Since all of the first j jobs with processing time p_j have a due date less than or equal to d_j, there cannot be a schedule that schedules exactly $I[i]$ early jobs with processing time $p^{(i)}$ if we consider only the first j jobs. Thus, we set $W_j[I,b,d] = \infty$. If $\sum_{i=1}^{\#p} I[i] \cdot p^{(i)} + b\Delta \leq d$, then we can schedule job j early.

The first possibility is to schedule job j in an already existing batch. Then the total weight of tardy jobs is $W_{j-1}[I_\ell, b, d]$.

The second possibility is to open a new batch for job j. Then we look at the entries $W_{j-1}[I_\ell, b-1, d']$ for $d' \leq d$.

The third possibility is to schedule j tardy. In this case, the weight is given by $W_{j-1}[I,b,d] + w_j$. Then the recursion for $W_j[I,b,d]$ is given by

$$W_j[I,b,d] = \min\{W_{j-1}[I_\ell,b,d], \min_{d' \leq d}\{W_{j-1}[I_\ell,b-1,d']\},\ W_{j-1}[I,b,d] + w_j\}.$$

Observe that the three cases correspond to the three options of adding j to a schedule of $\{1,\ldots,j-1\}$: (i) adding j to the last batch of early jobs, (ii) place j in a new batch on its own, (iii) scheduling j as tardy.

Correctness of our dynamic programming algorithm is immediate following the discussion above. The optimal schedule corresponds to the minimum entry $W_n[I,b,d]$ over all $I \in \{1,\ldots,n\}^{\#p}$, $b \in \{1,\ldots,n\}$, and $d \in \{0,d_1,\ldots,d_n\}$. Note that since table W_j has $n^{O(\#p)}$ entries, and computing each entry requires $O(\#d)$ time, computing the entire table can be done in time $n^{O(\#p)} \cdot O(\#d)$. Thus, the algorithm for computing all tables W_j runs in time $n^{O(\#p)} \cdot O(\#d)$, and the lemma follows. □

Now that we have shown how to solve $1|\text{s-batch}(\infty)|\sum w_jU_j$ efficiently for all instances with few distinct processing times, we next give an efficient algorithm for instances with few distinct weights.

Lemma 4 *Problem* $1|\text{s-batch}(\infty)|\sum w_jU_j$ *is solvable in time* $n^{O(\#w)} \cdot O(\#d)$.

Proof Let $J = \{1,\ldots,n\}$ denote our job set ordered according to EDD, and let $w^{(1)} < \cdots < w^{(\#w)}$ denote the different weights of all jobs in J. The algorithm is very similar to the algorithm in the proof of Lemma 3, except here we compute tables P_j that store minimum total processing time of early jobs, as opposed to minimum total weight of tardy jobs. Namely, for $I \in \{1,\ldots,n\}^{\#w}$, $b \in \{1,\ldots,n\}$, and $d \in \{0,d_1,\ldots,d_n\}$, entry $P_j[I,b,d]$ will equal the minimum total processing time of the early jobs in a schedule for jobs $\{1,\ldots,j\}$ that satisfies all properties required in the proof of Lemma 3, except that the second condition is rephrased to require exactly $I[i]$ early jobs, $i \in \{1,\ldots,\#w\}$, with weight $w^{(i)}$.

Fix $j \in \{1,\ldots,n\}$, and let $\ell \in \{1,\ldots,n\}$ be the index for which $w_j = w^{(\ell)}$. The base cases for computing $P_j[I,b,d]$ are very similar to those described in the proof of Lemma 3:

If both $P_{j-1}[I_\ell,b,d] + p_j > d$ and

$$\min_{d' \leq d}\{P_{j-1}[I_\ell,b-1,d'] + p_j + \Delta\} > d$$

hold or if $d > d_j$ then we cannot schedule exactly $I[i]$ jobs with weight $w^{(i)}$ early *including* job j if we consider only the first j jobs. Thus, we set $P_j[I,b,d] = P_{j-1}[I,b,d]$.

Otherwise, the main recursive formula is given by

$$P_j[I,b,d] = \min\{P_{j-1}[I_\ell,b,d] + p_j, \min_{d' \leq d}\{P_{j-1}[I_\ell,b-1,d'] + p_j + \Delta\},\ P_{j-1}[I,b,d]\}.$$

□

4 Serial batching with release dates

In this section, we show that the problem of minimizing the weighted number of tardy jobs with release dates on a single batch machine is W[1]-hard for parameters $\#p + \#r$ and $\#d + \#p$. That is, we prove Theorem 2. Thereafter, we give XP-algorithms for $1|\text{s-batch}(\infty), r_j|\sum w_jU_j$ parameterized by $\#p + \#w + \#r$, and parameterized by $\#d + \#p + \#w$. Note that if all jobs j have a common release date $r_j \equiv r$, then the problem $1|\text{s-batch}(\infty), r_j|\sum w_jU_j$ reduces to $1|\text{s-batch}(\infty)|\sum w_jU_j$ by releasing all jobs at time 0 and subtracting r_j from the due date d_j of each job; thus, the fixed-parameter algorithms from the preceding section apply.

We begin with parameter $\#p + \#r$; the hardness for parameter $\#d + \#p$ will follow almost immediately afterwards. To prove that $1|\text{s-batch}(\infty), r_j|\sum w_jU_j$ is W[1]-hard with respect to $\#p + \#r$, we present a reduction from the k-SUM problem. In this problem, we are given a set $\{x_1,\ldots,x_n\}$ of n positive integers, and a target integer t. The task is to decide if there exist k (not necessarily distinct) integers $x_{\pi(1)},\ldots,x_{\pi(k)} \in \{x_1,\ldots,x_n\}$ that sum up to t. Abboud et al. (2014) showed that k-SUM is W[1]-hard parameterized by k, even if all integers are in the range $\{1,\ldots,n^{ck}\}$ for some constant c.

The construction Let $(x_1,\ldots,x_n;t)$ be an instance of k-SUM, with $x_i \in \{1,\ldots,n^{ck}\}$ for each $i \in \{1,\ldots n\}$. Observe that due to their small range, each input integer x_i can be written in the form $x_i = \sum_{h=0}^{ck} \alpha_{i,h} \cdot n^h$ for integers $\alpha_{i,0},\ldots,\alpha_{i,ck} \in \{0,\ldots,n-1\}$, i.e., the *base n representation* of x_i. We will heavily exploit this property in our construction.

Write $X = \sum_i x_i$. Furthermore, we will assume throughout that $k-1$ times the largest integer in $\{x_1,\ldots,x_n\}$ is less than t. If this is not the case, one can slightly modify the input by adding kn^{ck} to each integer, and setting the target

to $t + k^2 n^{ck}$. (Note that this way it holds that $x_i \leq n^{c'k}$ for $c' = 3c$ for $i = 1, \ldots, n$, and thus the property of the base n representation still holds.) We construct an instance of $1|\text{s-batch}(\infty), r_j|\sum w_j U_j$ with $O(k)$ distinct processing times and release dates, such that there exists a feasible schedule with $\sum_j w_j U_j \leq kX - t + (n-1)k$, if and only if there exist k integers $x_{\pi(1)}, \ldots, x_{\pi(k)} \in \{x_1, \ldots, x_n\}$ that sum up to t:

- We create $(k-1)t$ identical jobs, called *leftover jobs*, each with the following parameters:
  - Processing time 1 and weight $k(X+n)$.
  - Release date 0 and due date $3kt$.
- For each $\ell \in \{1, \ldots, k\}$, and each input integer $x_i = \sum_{h=0}^{ck} \alpha_{i,h} \cdot n^h$, we create a set $J_{x(i),\ell}$ of *normal jobs* that corresponds to x_i. This set consists of $\alpha_{i,h}$ many jobs, for each $h \in \{0, \ldots, ck\}$, with the following parameters:
  - Processing time n^h and weight $n^h + n^h/x_i$.
  - Release date $r_\ell = (\ell-1)3t$ and due date $(\ell-1)3t + t + x_i$.
- We set the batch setup time to $\Delta = t$.
- We set the bound on the total weight of tardy jobs to $kX - t + (n-1)k$.

Observe that the total processing time of all jobs in the set $J_{x(i),\ell}$ is precisely x_i, and their total weight is $x_i + 1$. These two properties will be crucial later on. Also note that whereas the weights above are fractional, one can make them integral by multiplying them with $\prod x_i$. Under this multiplication, the encoding length of the numbers remains polynomial, as $\log(\prod x_i) = \sum_i \log(x_i) \leq n \max_i\{\log(x_i)\}$.

We now argue about the correctness of the reduction, by means of the subsequent two lemmas.

Lemma 5 *Suppose there exist $x_{\pi(1)}, \ldots, x_{\pi(k)} \in \{x_1, \ldots, x_n\}$ such that $\sum_i x_{\pi(i)} = t$. Then there exists a schedule with $\sum_j w_j U_j \leq kX - t + (n-1)k$.*

Proof We create a schedule with $2k+1$ batches $B_1, \ldots, B_{2k+1}$. For $\ell \in \{1, \ldots, k\}$, we schedule all jobs in set $J_{\pi(\ell),\ell}$ in batch $B_{2\ell-1}$, and $t - x_{\pi(\ell)}$ leftover jobs in batch $B_{2\ell}$. The starting time of batch $B_{2\ell-1}$ is at $3t(\ell-1)$, and that of batch $B_{2\ell}$ is at $3t(\ell-1) + t + x_{\pi(\ell)}$. The remaining jobs are all scheduled in batch B_{2k+1} which starts at time $3kt$. Note that in this way all jobs are scheduled after their release date, and only jobs in the last batch B_{2k+1} are tardy. An easy calculation shows that the total weight of jobs in this last batch is

$$\sum_{j \in B_{2k+1}} w_j = kX + kn - \sum_{i=1}^{k} (x_{\pi(i)} + 1)$$
$$= kX - t + (n-1)k.$$

We illustrate Lemma 5 by an example in Fig. 2.

The converse of Lemma 5 requires more technical detail. We therefore introduce some further notation that will be used throughout the remainder of the section. Assume our constructed instance of $1|\text{s-batch}(\infty), r_j|\sum w_j U_j$ admits a schedule where the total weight of tardy jobs is at most $kX + (n-1)k - t$. Let $B_1, \ldots, B_b, B_{b+1}$ denote the batches of this schedule, with respective starting times $S_1 < \cdots < S_{b+1}$ and completion times $C_1 < \cdots < C_{b+1}$. Below we modify this schedule, without increasing the total weight of tardy jobs, in order to make our arguments easier.

Lemma 6 *Suppose that the constructed instance of $1|\text{s-batch}(\infty), r_j|\sum w_j U_j$ has a schedule where the total weight of tardy jobs is at most $kX + (n-1)k - t$. Then it has a schedule where the total weight of tardy jobs is at most $kX + (n-1)k - t$ and with batches $B_1, \ldots, B_b, B_{b+1}$, scheduled in that order, where:*

- *All tardy jobs are in B_{b+1}, and include no leftover jobs.*
- *All early jobs are in $B_1, \ldots, B_b$, and include normal jobs with total weight at least $t + k$.*

Proof Consider any schedule where the total weight of tardy jobs is at most $kX + (n-1)k - t$ and with batches $B_1, \ldots, B_b$, scheduled in that order, that has at most $kX - t + (n-1)k$ total weight of tardy jobs. We first observe that no leftover job is tardy, as a single leftover job has weight $k(X+n) > kX - t + (n-1)k$. Moreover, as the total weight of all normal jobs of the instance is $k(X+n)$, the total weight of the early normal jobs must be at least $t + k$. Finally, we can move all tardy jobs to a new batch B_{b+1} that starts right after B_b completes, deleting all empty batches resulting from this, without increasing the total weight of tardy jobs. □

Due to Lemma 6, some normal jobs must be early. For $\ell \in \{1, \ldots, k\}$, we use E_ℓ denote the early jobs of type ℓ in the schedule; here, a job has *type* ℓ if it belongs to $J_{x(i),\ell}$ for some $i \in \{1, \ldots, n\}$. Then $\bigcup_\ell E_\ell \neq \emptyset$. We use $p(E_\ell)$ and $w(E_\ell)$ to respectively denote the total processing time and weight of jobs in E_ℓ, i.e., $p(E_\ell) = \sum_{j \in E_\ell} p_j$ and $w(E_\ell) = \sum_{j \in E_\ell} w_j$.

Lemma 7 *For each $\ell \in \{1, \ldots, k\}$ with $E_\ell \neq \emptyset$, there is a unique batch $B(\ell) \in \{B_1, \ldots, B_b\}$ with $E_\ell \subseteq B$.*

Proof Choose some non-empty E_ℓ. Then each job $j \in E_\ell$ is released at time r_ℓ and has a due date of $r_\ell + t + x < r_\ell + 2t$ for some $x \in \{x_1, \ldots, x_n\}$ (the inequality follows, as all x_i are smaller than t). As batch setup requires t units of time, and all jobs in E_ℓ are early, there must be some batch that contains all jobs of E_ℓ. Furthermore, this batch cannot contain jobs of some $E_{\ell'}$, $\ell' \neq \ell$, since those jobs either have due dates prior to r_ℓ (in case $\ell' < \ell$), or release dates that are later than the due dates of jobs in E_ℓ (in case $\ell' > \ell$). □

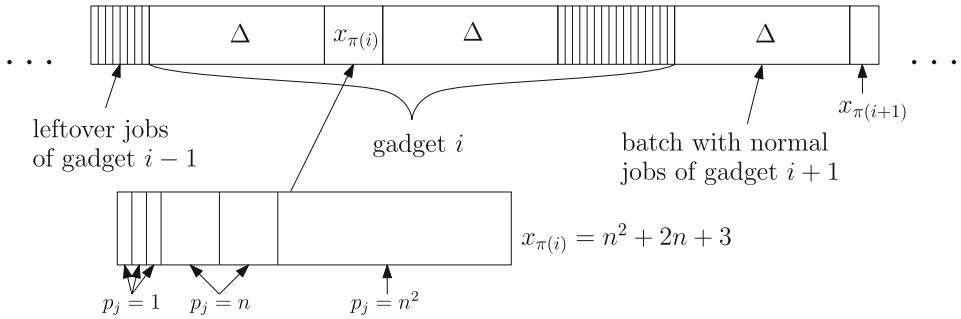

Fig. 2 An illustration of how the schedule given in Lemma 5 looks like

Lemma 7 implies that we can assume there is a specific batch associated with each non-empty E_ℓ. Let d_ℓ be the earliest due date in E_ℓ. Then $d_\ell = (\ell-1)3t + t + x_{\pi(\ell)}$ for some integer $x_{\pi(\ell)} \in \{x_1, \ldots, x_n\}$. Thus, there is also a specific due date and input integer associated with E_ℓ.

Lemma 8 *For each $\ell \in \{1, \ldots, k\}$ with $E_\ell \neq \emptyset$ we have:*

- $p(E_\ell) \leq x_{\pi(\ell)}$.
- $w(E_\ell) \leq p(E_\ell) + 1$, *and this holds with equality if and only if $E_\ell = J_{\pi(\ell),\ell}$.*

Proof According to Lemma 7, there is a unique batch $B(\ell)$ which includes all jobs of E_ℓ. Let $p(B(\ell))$ denote the sum of processing times of jobs in $B(\ell)$. As the release date of all jobs in E_ℓ is $r_\ell = (\ell-1)\cdot 3t$, and the setup time of $B(\ell)$ is t, it must be that $p(E_\ell) \leq p(B(\ell)) \leq x_{\pi(\ell)}$; otherwise, jobs in E_ℓ with due date d_ℓ would be late. Now, for each job $j \in E_\ell$, let $x(j) \in \{x_1, \ldots, x_n\}$ denote the integer to which j is associated (i.e., $j \in J_{x(j),\ell}$). Then $x(j) \geq x_{\pi(\ell)}$, by definition of $x_{\pi(\ell)}$. Since $p(E_\ell) \leq x_{\pi(\ell)}$, we have

$$\begin{aligned} w(E_\ell) &= \sum_{j \in E_\ell} (p_j + p_j/x(j)) \\ &\leq \sum_{j \in E_\ell} (p_j + p_j/x_{\pi(\ell)}) \\ &= p(E_\ell) + p(E_\ell)/x_{\pi(\ell)} \\ &\leq p(E_\ell) + 1. \end{aligned}$$

Note that the first inequality is strict if and only if there is a job $j \in E_\ell \backslash J_{\pi(\ell),\ell}$ as $x(j) \geq x_{\pi(\ell)}$ and the second inequality is strict if and only if $p(E_\ell) < x_{\pi(\ell)}$. Hence, equality holds if and only if $E_\ell = J_{\pi(\ell),\ell}$. The statement of the lemma thus follows. □

Lemma 9 *$E_\ell \neq \emptyset$ for each $\ell \in \{1, \ldots, k\}$.*

Proof By Lemma 6, we have $t + k \leq \sum_\ell w(E_\ell)$. By Lemma 8, we have $p(E_\ell) \leq x_{\pi(\ell)}$ and $w(E_\ell) \leq p(E_\ell) + 1$. Thus,

$$t \leq \sum_{\ell=1}^{k} w(E_\ell) - k \leq \sum_{\ell=1}^{k} p(E_\ell) \leq \sum_{\ell=1}^{k} x_{\pi(\ell)},$$

where $x_{\pi(\ell)} = 0$ if $E_\ell = \emptyset$ in the summation above. Since any $k-1$ integers in $\{x_1, \ldots, x_n\}$ sum up to a number which is smaller than t, it must be that $x_{\pi(\ell)} > 0$ for all $\ell \in \{1, \ldots, k\}$, and the statement of the lemma follows. □

Lemma 10 *If there is a schedule where the total weight of tardy jobs is at most $kX + (n-1)k - t$, then there is one with batches $B_1, \ldots, B_{2k+1}$ scheduled in that order, where for each $\ell \in \{1, \ldots, k\}$:*

- *$B_{2\ell-1}$ is scheduled at time $3t(\ell-1)$; $B_{2\ell}$ is scheduled immediately after $B_{2\ell-1}$ is completed.*
- *$B_{2\ell-1}$ contains only normal jobs of type of ℓ, and $B_{2\ell}$ contains only leftover jobs.*
- *All tardy jobs are in B_{2k+1}, and are normal.*

Proof Let $B_1, \ldots, B_{b+1}$ be the batches of our schedule as in Lemma 6. We modify the batches of this schedule so as to fit the requirements of the lemma without increasing the total weight of tardy jobs in the schedule.

We first note that for each ℓ, batch $B(\ell)$ is completely processed in the interval $[3t(\ell-1), 3t(\ell-1)+2t]$. Thus, if there is no batch between $B(\ell)$ and $B(\ell+1)$, we might as well add one as the time between the completion time of $B(\ell)$ and the starting time of $B(\ell+1)$ is at least t. Since there is a batch $B(\ell)$ for each $\ell \leq k$, by Lemma 9, there are $2k$ batches consisting only of early jobs.

Suppose that the completion time of a batch B_i is in the interval $(3t(\ell-1), 3t(\ell-1)+t]$ for some ℓ. Then B_i cannot contain type ℓ jobs, as it started before $3t(\ell-1)$. Hence, B_i only contains leftover jobs. We can move some leftover jobs from B_i to $B_{i+1} = B(\ell)$ and simultaneously reduce the starting time of $B(\ell)$ by the number of moved jobs, until the starting time of B_i equals $3t(\ell-1)$.

If no batch is completed in $(3t(\ell-1), 3t(\ell-1)+t]$, then we can start $B(\ell)$ at time $3t(\ell-1)$. This can only decrease the completion times of the jobs. If there are leftover jobs in batch $B(\ell) = B_{2\ell-1}$, then we can move them to batch $B_{2\ell}$. They will not be late, as the completion time of $B_{2\ell}$ is at most $3kt$. □

Lemma 11 *If the constructed instance of* $1|s\text{-}batch(\infty), r_j|\sum w_jU_j$ *admits a schedule with* $\sum_j w_jU_j \leq kX - t + (n-1)k$*, then there are* $x_{\pi(1)}, \ldots, x_{\pi(k)} \in \{x_1, \ldots, x_n\}$ *with* $\sum_i x_{\pi(i)} = t$.

Proof Let $B_1, \ldots, B_{2k+1}$ be the batches of a schedule as promised by Lemma 10 for our 1|s-batch$(\infty), r_j|\sum w_jU_j$ instance with $\sum_j w_jU_j \leq kX - t + (n-1)k$. Then batch B_{2k} completes at time $C_{2k} \leq 3kt$, since $3kt$ is the latest due date of the input jobs. Since there are $2k$ batches with early jobs, and the setup time for each of these batches is t, we have $\sum_{\ell=1}^{2k} p(B_\ell) \leq kt$. Thus, as the total processing times of all leftover jobs is $(k-1)t$, we have

$$\sum_{\ell=1}^{k} p(E_\ell) = \sum_{\ell=1}^{2k} p(B_\ell) - \sum_{\ell=1}^{k} p(B_{2\ell}) = \sum_{\ell=1}^{2k} p(B_\ell) - (k-1)t \leq t.$$

Recall that, by Lemmas 6 and 7, we also have

$$t \leq \sum_{\ell=1}^{k} w(E_\ell) - k \leq \sum_{\ell=1}^{k} p(E_\ell).$$

Therefore, $\sum p(E_\ell) = t$ and $\sum w(E_\ell) = \sum p(E_\ell) + k$. The latter equality can only happen if $w(E_\ell) = p(E_\ell) + 1$ for all $\ell = 1, \ldots, k$, which in turn implies by Lemma 9 that $p(E_\ell) = x_{\pi(\ell)}$ for all $\ell = 1, \ldots, k$. So $\sum x_{\pi(\ell)} = t$, and the claim follows. □

4.1 Parameter #d + #p

Lemmas 5 and 11 combined prove that our construction indeed shows W[1]-hardness for parameter $\#p+\#r$. We next show that this construction can be transformed to show hardness for parameter $\#p+\#d$ (formalizing the intuitive notion of time reversibility).

Lemma 12 *For non-negative integers* k, k'*, any instance of* $1|s\text{-}batch(\infty), r_j|\sum w_jU_j$ *with* k *distinct release dates and* k' *distinct due dates can be transformed into an instance of* $1|s\text{-}batch(\infty), r_j|\sum w_jU_j$ *with* k' *distinct release dates and* k *distinct due dates, which has the same objective value.*

Proof Let J be a set of n jobs forming an instance of 1|s-batch$(\infty), r_j|\sum w_jU_j$. We create a set J' of n jobs, as follows. For each job $j \in J$, we create one job $j' \in J'$ with $p_{j'} = p_j$, $w_{j'} = w_j$, $r_{j'} = -d_j$ and $d_{j'} = -r_j$. Observe that the problem of finding a maximum-weight set of early jobs is the same for both J and J':

Let σ be a schedule for J, and let $J_e(\sigma)$ be the set of jobs in J that are early in σ. For $j \in J_e(\sigma)$ let S_j denote its starting time of j and C_j its completion time. Then we obtain a schedule σ' for J' by setting the start time of j' to be $S_{j'} = -C_j$ for all jobs $j \in J_e(\sigma)$ and scheduling the remaining jobs late. No two jobs will be processed at the same time, as the intervals $(S_j, C_j), (S_{j'}, C_{j'})$ are pairwise disjoint for all $j, j' \in J_e(\sigma)$. Thus the intervals $(-C_j, S_j), (-C_{j'}, S_{j'})$ are also pairwise disjoint for all $j, j' \in J'_e(\sigma')$. Further, for each $j \in J_e(\sigma)$ we have $S_j \geq r_j$ and $d_j \geq C_j$ and thus also $r_{j'} = -d_j \leq -C_j = S_{j'}$ and $d_{j'} = -r_j \geq -S_j = C_{j'}$.

Similarly, given the set $J'_e(\sigma')$ of early jobs for a schedule σ' for J' we obtain a schedule for J such that all jobs j for which $j' \in J'_e(\sigma')$ are scheduled early, by setting $S_j = -C_{j'}$.

This shows that the problem 1|s-batch$(\infty), r_j|\sum w_jU_j$ with parameter $\#d + \#p$ is as hard as 1|s-batch$(\infty), r_j|\sum w_jU_j$ with parameter $\#p + \#r$. □

Corollary 1 *Problem* $1|s\text{-}batch(\infty), r_j|\sum w_jU_j$ *is* W[1]*-hard for parameter* $\#d + \#p$.

4.2 XP algorithms

Last in this section, we give an XP-algorithm for the problem 1|s-batch$(\infty), r_j|\sum w_jU_j$ parameterized by $\#p+\#r+\#w$. We use the following notation: Similarly to the due dates, we order the release dates as follows: $r^{(1)} < r^{(2)} \cdots < r^{(\#d)}$.

Lemma 13 *Problem* $1|s\text{-}batch(\infty), r_j|\sum w_jU_j$ *is solvable in time* $n^{f(\#p+\#r+\#w)}$.

Proof Let I be the set of job types with respect to processing time, weight and release date. Let n_i be the number of jobs of type i. Let $U = \{v \in \{1, \ldots, n\}^I\}$, and let $V = \{v = (v_1, \ldots, v_{\#r}) \in U^{\#r} \mid \sum_{\ell=1}^{\#r}(v_\ell)_i \leq n_i\}$ denote the space of possible solution vectors. For each element $v \in V$ we decide if one can obtain a schedule that starts $(v_\ell)_i$ early jobs of type $i \in I$ in the interval $[r^{(\ell)}, r^{(\ell+1)})$.

First, note that if such a schedule exists then we can assume that jobs of types i are scheduled in order of their due date, and only the $\sum_{\ell=1}^{\#r}(v_\ell)_i$ jobs of type i with the latest due dates are scheduled early. Hence we know which jobs are started in each interval $[r^{(\ell)}, r^{(\ell+1)})$.

Second, notice that if we schedule the jobs that are started in $[r^{(\ell)}, r^{(\ell+1)})$ in (EDD)-order starting new batches only if it is necessary, i.e. if adding the job to the current batch would cause another job in the batch to be tardy. Then we also get a schedule for these jobs that ends as early as possible. Thus all we need to do in order to decide whether such a schedule exists is to the following: First schedule all jobs that start in $[r^{(1)}, r^{(2)})$ in (EDD). Then let t_1 be the date where the last of these jobs is finished. Then we schedule all jobs that start in $[r^{(2)}, r^{(3)})$ in (EDD) but the starting time of the first batch is $\max\{r^{(2)}, t_1\}$. Then let t_2 be the date where the last of

these jobs is finished. We then continue in the obvious way. If all jobs scheduled are early and no job is started before its release date then there is such a schedule; otherwise, no such schedule exists. From all schedules we obtain, we take the one that maximizes $\sum_{\ell,i}(v_\ell)_i w_i$. The total run time is $n^{O(\#p+\#r^2+\#w)}$. □

Using Lemma 12 we also get the following result:

Corollary 2 *Problem* $1|s\text{-}batch(\infty), r_j|\sum w_j U_j$ *is solvable in* $n^{f(\#d+\#p+\#w)}$ *time.*

5 Serial batching with bounded batch sizes

In this section, we consider serial batching where batches have bounded size $|B| \leq b$; this yields problem $1|\text{s-batch}(b)|\sum w_j U_j$.

We will use the notion of job types: Each job $j \in J$ has a *type*, which is given by the vector $\tau(j) = (p_j, w_j, d_j, r_j)$. In some settings parts of the tuple can be omitted, which allows us to shortcut the job type. For example, a job of type (p_j, w_j, d_j, r_j) is also of type (p_j, d_j, r_j). We denote the set of all job types of an instance $\mathcal{I}$ by $\mathcal{T}$. For each type $\tau \in \mathcal{T}$ let n_τ be the number of jobs in $\mathcal{I}$ of type τ; further, let d_τ, p_τ, r_τ and w_τ denote the due date, processing time, release date and weight of jobs with type τ.

First, we show that $1|\text{s-batch}(b)|\sum w_j U_j$ is fixed-parameter tractable for parameter $\#d + \#w$, proving the first part of Theorem 3.

Lemma 14 *Problem* $1|s\text{-}batch(b)|\sum w_j U_j$ *can be solved in time* $f(\#d + \#w) \cdot n^{O(1)}$.

Proof For an instance $\mathcal{I}$ of $1|\text{s-batch}(b)|\sum w_j U_j$, we set up a mixed-integer linear program (MILP) to find an optimal schedule. The variables of the MILP fall into one of three classes.

For the first class, let $I = \{(w, d) \mid (w, p, d) \in \mathcal{T} \text{ for some } p\}$ be the set of job types with respect to weight and due date. For each type $i \in I$ and each $\ell \in \{1, \ldots, \#d\}$ we have one integer variable $x_{(i,\ell)}$ to indicate the number of jobs of type i which finish in the time interval $(d^{(\ell-1)}, d^{(\ell)}]$ in the computed schedule. (Note that this means that their batches finish in the interval.)

In the second class, for each job type $\tau = (d_\tau, p_\tau, w_\tau) \in \mathcal{T}$ and each $\ell \in \{1, \ldots, \#d\}$ we have one fractional variable $y_{(\tau,\ell)} \in [0, n_\tau]$ to indicate the number jobs of type τ which are processed in time before their due date $d^{(\ell)}$. (Recall that n_τ is the number of jobs of type τ.)

In the third class, for each index $\ell \in \{1, \ldots, \#d\}$ we have one integer variable z_ℓ to indicate the number of batches that are completed before or at time $d^{(\ell)}$. Finally, set $z_0 = 0$. The MILP is given by

$$\min \sum_{\tau \in \mathcal{T}} (n_\tau - y_{(\tau,\#d)}) w_\tau \tag{1}$$

$$z_\ell \geq z_{\ell-1} + \frac{1}{b}\sum_{i \in I} x_{(i,\ell)}, \qquad \ell = 1, \ldots, \#d, \tag{2}$$

$$\sum_{\ell_0 \leq \ell} x_{(i,\ell_0)} = \sum_{\substack{\tau \in \mathcal{T}, \\ w_\tau = w_i \wedge d_\tau = d_i}} y_{(\tau,\ell)}, \quad i \in I, \ell = 1, \ldots, \#d, \tag{3}$$

$$z_\ell \Delta + \sum_{\tau \in \mathcal{T}} p_\tau y_{(\tau,\ell)} \leq d^{(\ell)} \qquad \ell = 1, \ldots, \#d. \tag{4}$$

The MILP has $\#d(|I| + 1) = O(\#d^2 \cdot \#w)$ integer variables and $|\mathcal{T}|\#d = O(|\mathcal{T}|^2)$ fractional variables. It can be solved by Lenstra's algorithm (Lenstra, 1983) for integer programming in fixed dimension in time $f(\#d + \#w) \cdot n^{O(1)}$.

It remains to show that optimal solutions of value W to the MILP correspond to optimal schedules with weighted number of tardy jobs equal to W. A key observation is that, given an optimal solution to the MILP, we can assume that all variables $y_{(\tau,\ell)}$ take integer values. This is due to the fact that, given a job type $\tau \in \mathcal{T}$ and an index $\ell \in \{1, \ldots, \#d\}$, we can assume that if $y_{(\tau,\ell)} < n_j$ then $y_{(\tau',\ell)} = 0$ for all τ' with $p_{\tau'} > p_\tau$, $w_{\tau'} = w_\tau$ and $d_{\tau'} = d_\tau$. For if that was not the case, then we can increase $y_{(\tau,\ell)}$ and decrease $y_{(\tau',\ell)}$ by the same amount, without changing the objective value or violating constraint (3) or constraint (4). The intuition here is that we can process jobs of type $i \in I$ in increasing order of their processing time.

Note that (2) assures that we use $\lceil \frac{1}{b}\sum_{i \in I} x_{(i,\ell)} \rceil$ batches ending in $(d^{(\ell-1)}, d^{(\ell)}]$ which is the minimum number of batches needed to complete all jobs ending in that interval. Constraint (3) is for determining the exact types of the jobs that are processed rather than just the type with respect to weight and due date. As said, we can assume that y-variables are integral in an optimum solution. Constraint (4) ensures that all early jobs are indeed completed before their due date.

To obtain a schedule from a solution to the MILP, we first process $x_{(i,1)}$ jobs of type i for each i in order of their processing times with ties broken arbitrarily, and always starting a new batch when necessary and closing the last batch at the end. Then we can continue with $x_{(i,2)}$ jobs of type i for each i the same way, and so on. Conversely, a schedule translates into a solution (also fulfilling (2)) using the interpretations for the variables. □

If $\#d + \#p$ (rather than $\#d + \#w$) is our parameter, then we obtain fixed-parameter tractability even in the presence of release dates. More precisely, we can solve instances where jobs additionally have different release dates as long as the number of different release dates is our parameter.

Lemma 15 *Problem* $1|s\text{-}batch(b), r_j|\sum w_j U_j$ *can be solved in time* $f(\#d + \#p + \#r) \cdot n^{O(1)}$.

Proof Let $T = \{r_j \mid j \in J\} \cup \{d_j \mid j \in J\}$ be the set of critical time points. Further, we order $T = \{t_1, \ldots, t_k\}$ increasingly, i.e., $t_1 < \cdots < t_k$. We again design a MILP, but this time with slightly more variables than the MILP in the proof of 14. Again, there are three classes of variables.

For the first class, we set $I = \{(p, r, d) \mid (p, w, r, d) \in \mathcal{T}$ for some w$\}$ to be the set of job types with respect to weight and due date. Then instead of variables $x_{(i,\ell)}$, we have integer variables $x_{(i,\ell,\ell')}$ to indicate the number of early jobs of a given type that are processed in batches starting at or after t_ℓ but before $t_{\ell+1}$ and completed before or at $t_{\ell'}$ but after $t_{\ell'-1}$. Note that we do not create variables $x_{(i,\ell,\ell')}$ if $d_i < t_{\ell'}$ or $r_i > t_\ell$.

For the second class, we use variables y_τ to indicate the number of early jobs of type $\tau \in \mathcal{T}$ in the computed schedule.

For the third class, instead of variables z_ℓ, this time we will use integer variables $z_{(\ell,\ell')}$ for any $\ell < \ell'$ to indicate the number of batches that start at or after t_ℓ but before $t_{\ell+1}$ and finish before or at $t_{\ell'}$ but after $t_{\ell'-1}$.

The MILP has as objective function

$$\min \sum_{\tau \in \mathcal{T}} (n_\tau - y_\tau) w_\tau,$$

and the following four kinds of constraints:

$$\sum_{i \in I} x_{(i,\ell,\ell')} \leq b z_{(\ell,\ell')} \text{ for any } 1 \leq \ell < \ell' \leq k$$

$$\sum_{\ell,\ell'} x_{(i,\ell,\ell')} = \sum_{\substack{\tau \in \mathcal{T}, \\ p_\tau = p_i \wedge r_i = r_\tau \wedge d_\tau = d_i}} y_\tau \text{ for each } i \in I$$

$$t + \sum_{t \leq t_\ell < t_{\ell'} \leq t'} \left(z_{(\ell,\ell')} \Delta + \sum_{i \in I} p_i x_{(i,\ell,\ell')} \right) \leq t'$$
$$\text{for each each } t, t' \in T \text{ with } t < t'$$

$$y_\tau \leq n_\tau \text{ for each } \tau \in \mathcal{T}.$$

We need two more kinds of constraints to guarantee that if there is a long batch, i.e., a batch that starts before t_ℓ and ends at or before $t_{\ell'}$ but after $t_{\ell'-1} \geq t_\ell$, then there cannot be any other batch starting and ending in $[t_j, t_{j'}]$ for any pair $(j, j') \in \{\ell, \ldots, \ell'\}^2 \setminus \{(t_\ell, t_{\ell+1}), (t_{\ell'-1}, t_{\ell'})\}$. These constraints are thus:

$$z_{(\ell_1,\ell_2)} + z_{(\ell_3,\ell_4)} \leq 1 \qquad \text{if } \ell_1 \leq \ell_3 < \ell_3 + 2 \leq \ell_4 \leq \ell_2 \tag{5}$$

$$\frac{z_{(\ell,\ell+1)}}{n} + z_{(\ell_1,\ell_2)} \leq 1 \qquad \text{if } \ell_1 < \ell \text{ and } \ell_2 > \ell + 1. \tag{6}$$

Using the interpretations of the variables, from a given schedule one can easily construct a feasible solution of the MILP with same value. In an optimal solution of the MILP, all variables of the form y_τ are integral, and $y_\tau \leq n_\tau$ implies $y_{\tau'} = 0$ for all other types τ' with the same processing time, release date and due date but higher weight. To see this, assume there is some non-integral y_τ. Let τ be of (sub)type $i \in I$. As

$$\sum_{\tau \in \mathcal{T}, w_\tau = w_i \wedge r_i = r_\tau \wedge d_\tau = d_i} y_\tau = \sum_{\ell,\ell'} x_{(i,\ell,\ell')}$$

is integral, there is another non-integral variable $y_{\tau'}$ so that τ' is also of type i. Assume, without loss of generality, that $w_\tau > w_{\tau'}$. As n_τ is integral, it holds $y_\tau < n_\tau$. Thus we can increase y_τ and decrease $y_{\tau'}$ by the same amount until either $y_\tau = n_\tau$ or $y_{\tau'} = 0$. The solution we get is still feasible but its value is smaller, contradicting the optimality of our previous solution. The same argumentation can be used to show that $y_\tau \leq n_\tau$ implies $y_{\tau'} = 0$ for all other types τ' with the same processing time, release date and due date but higher weight.

Now to create a schedule we create $z_{(\ell,\ell')}$ batches $\mathcal{B}_{\ell,\ell'}$ for each variable $z_{(\ell,\ell')}$ and fill them with appropriate jobs, i.e., such that there are $x_{(i,\ell,\ell')}$ jobs of type i assigned to them. We schedule the batches as follows: If batch B is in $\mathcal{B}_{\ell,\ell'}$ and batch B' is in $\mathcal{B}_{\ell_1,\ell_2}$, then we schedule B before B' if $\ell < \ell_1$ or if $\ell = \ell_1$ and $\ell' < \ell_2$. Apart from this rule, batches can be scheduled in arbitrary order. Given this ordering, we schedule batch $B \in \mathcal{B}_{\ell,\ell'}$ at the completion time of the previous batch if it finishes later than t_ℓ, or at t_ℓ otherwise.

We need to show that indeed all $\sum_{\ell,\ell'} x_{(i,\ell,\ell')}$ jobs of type i scheduled in these kind of batches are early for each type i. Suppose, for sake of contradiction, that there is late job j in batch $B \in \mathcal{B}_{\ell,\ell'}$ for some ℓ and ℓ'. Let t_{ℓ_0} be the latest time point less than or equal to t_ℓ such that there is idle time before t_{ℓ_0}, or—if no such time point exists—we set t_{ℓ_0} to be the smallest release date.

We claim that

$$t_{\ell_0} + \sum_{\ell_0 \leq \ell_1, \ell_2 \leq \ell'} \left(z_{(\ell_1,\ell_2)} \Delta + \sum_{i \in I} p_i x_{(i,\ell_1,\ell_2)} \right) > t_{\ell'}.$$

To see that, notice that only jobs in batches in $\mathcal{B}_{\ell_1,\ell_2}$ with $\ell_1 \geq \ell_0$ and $\ell_2 \leq \ell'$ are scheduled before the completion time of j. This holds true as any batch $B' \in \mathcal{B}_{\ell_1,\ell_2}$ with $\ell_1 < \ell_0$ is completed before t_{ℓ_0} by definition of ℓ_0 and any batch $B' \in \mathcal{B}_{\ell_1,\ell_2}$ with $\ell_1 \geq \ell$ and $\ell_2 > \ell'$ is scheduled later than j. Further, we have $z_{(\ell_1,\ell_2)} = 0$ if $\ell_1 < \ell$ and $\ell_2 > \ell'$ by constraint (5) and (6) using that $z_{(\ell,\ell')} \geq 1$. However, our claim contradicts the feasibility of such a schedule; thus, j cannot be late. □

6 Discussion and open problems

We provided an extensive multivariate analysis of the single-machine batch scheduling problem to minimize the weighted number of tardy jobs. In particular, we significantly refined and extended the work of Hochbaum and Landy (1994), as well as Hermelin et al. (2018).

Several open questions remain, even for the non-batch setting. It appears especially challenging to answer if $1||\sum w_jU_j$ is fixed-parameter tractable for parameter $\#p$ or parameter $\#w$, or is W[1]-hard for either of those parameterizations. This question was already stated by Hermelin et al. (2018). Naturally, we do not know the answer to this question for the more general $1|\text{s-batch}(\infty)|\sum U_jw_j$ problem; however, we also do not know the status of parameter $\#p + \#w$ for which $1||\sum w_jU_j$ is known to be fixed-parameter tractable (Hermelin et al., 2018). An interesting question is if $1|\text{s-batch}(b)|\sum U_j$ is fixed-parameter tractable for parameter $\#p$ or b, or even solvable in polynomial time.

Another interesting research direction would be to explore the parameterized complexity of parallel batching problems.

Funding M.M. and S.O. were partially supported by Deutsche Forschungsgemeinschaft, DFG Grant MN 59/4-1.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

Abboud, A., Lewi, K., & Williams, R. (2014). Losing weight by gaining edges. In *Proceedings of the ESA 2014. Lecture Notes Computer Science* (Vol. 8737, pp. 1–12).

Baptiste, P. (2000). Batching identical jobs. *Mathematical Methods of Operational Research, 52*(3), 355–367.

Brucker, P. (2007). *Scheduling* (5th ed.). Berlin: Springer Berlin Heidelberg.

Brucker, P., & Kovalyov, M. Y. (1996). Single machine batch scheduling to minimize the weighted number of late jobs. *Mathematical Methods of Operational Research, 43*(1), 1–8.

Cheng, T. C. E., & Kovalyov, M. Y. (2001). Single machine batch scheduling with sequential job processing. *IIE Transactions, 33*(5), 413–420.

Cygan, M., Fomin, F. V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., & Saurabh, S. (2015). *Parameterized algorithms*. Cham: Springer.

Etscheid, M., Kratsch, S., Mnich, M., & Röglin, H. (2017). Polynomial kernels for weighted problems. *Journal of Computer and System Sciences, 84*, 1–10.

Fowler, J. W., & Mönch, L. (2022). A survey of scheduling with parallel batch (p-batch) processing. *European Journal of Operational Research, 298*(1), 1–24.

Goemans, M. X., & Rothvoss, T. (2020). Polynomiality for bin packing with a constant number of item types. *Journal of the ACM, 67*(6), 1–21.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnoy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 3*, 287–326.

Hermelin, D., Karhi, S., Pinedo, M., & Shabtay, D. (2018). New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Annals of Operations Research, 298*, 271–287.

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

Hochbaum, D. S., & Landy, D. (1994). Scheduling with batching: minimizing the weighted number of tardy jobs. *Operations Research Letters, 16*(2), 79–86.

Karp, R. M. (1972). *Reducibility among combinatorial problems* (pp. 85–103). Springer.

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

Knop, D., Koutecký, M., & Mnich, M. (2020). Combinatorial *n*-fold integer programming and applications. *Mathematical Programming, 184*(1–2), 1–34.

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

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

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

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

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

Pinedo, M. (2016). *Scheduling: Theory, algorithms, and systems* (5th ed.). Cham: Springer.

Potts, C. N., & Kovalyov, M. Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research, 120*(2), 228–249.

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

van Bevern, R., Bredereck, R., Bulteau, L., Komusiewicz, C., Talmon, N., & Woeginger, G. J. (2016). Precedence-constrained scheduling problems parameterized by partial order width. In *Proceedings of the DOOR 2016. Lecture Notes Computer Science* (Vol. 9869, pp. 105–120).

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

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

Webster, S., & Baker, K. R. (1995). Scheduling groups of jobs on a single machine. *Operations Research, 43*(4), 692–703.