European Journal of Operational Research 286 (2020) 129–137

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A common approximation framework for early work, late work, and resource leveling problems☆

Péter Györgyi, Tamás Kis*

Institute for Computer Science and Control, Kende str 13–17, Budapest 1111, Hungary

ARTICLE INFO

Article history:
Received 16 September 2019
Accepted 9 March 2020
Available online 18 March 2020

Keywords:
Scheduling
late work minimization
early work maximization
resource leveling
approximation algorithms

ABSTRACT

We study the approximability of four scheduling problems on identical parallel machines. In the *late work minimization problem*, the jobs have arbitrary processing times and a common due date, and the objective is to minimize the *late work*, defined as the sum of the portion of the jobs done after the due date. A related problem is the maximization of the *early work*, defined as the sum of the portion of the jobs done before the due date. We describe a polynomial time approximation scheme for the early work maximization problem, and we extended it to the late work minimization problem after shifting the objective function by a positive value that depends on the problem data. We also prove an inapproximability result for the latter problem if the objective function is shifted by a constant which does not depend on the input. These results remain valid even if the number of the jobs assigned to the same machine is bounded. This leads to an extension of our approximation scheme to two variants of the resource leveling problem with unit time jobs, for which no approximation algorithm is known.

1. Introduction

Late work minimization, introduced by the pioneering paper of Błażewicz (1984), is an important area of machine scheduling, for an overview see Sterna (2011). The variant we are going to study in this paper can be briefly stated as follows. We have identical parallel machines and a set of jobs with arbitrary processing times, and a common due date. We seek a schedule which minimizes the sum of the portion of the jobs done after the due date. A strongly related problem is the maximization of the early work, where we have the same data and the objective is to maximize the sum of the portion of the jobs done before the common due date. However, the list of the results for maximizing the early work is much shorter than that for the late work minimization problem, see e.g., Sterna and Czerniachowska (2017), Chen, Liang, Sterna, Wang, and Błażewicz (2020b).

The applications of the late work optimization criterion range from modeling the loss of information in computational tasks to the measurement of dissatisfaction of the customers of a manufacturing company. In particular, Błażewicz (1984) studies a parallel processor scheduling problem with preemptive jobs where each job processes some samples of data (or measurement points), and if the processing completes after the job's due date, then it causes a loss of information. A natural objective is to minimize the information loss, which is equivalent to the minimization of the total late work. A small flexible manufacturing system is described in Sterna (2007), where the application of the late work criterion is motivated by the interests of the customers as well as by that of the owner of the system. The common interest of the customers is to have the portions of their orders finished after the due date minimized. In turn, for the owner of the system, the amount of late work is a measure of dissatisfaction of the customers. As for early work maximization, we can adapt the same examples considering gain and satisfaction instead of loss and dissatisfaction, respectively.

We have three major sources of motivation for studying the approximability of the early work maximization, and the late work minimization problems:

(i) Chen, Sterna, Han, and Błażewicz (2016) establish the complexity of late work minimization in a parallel machine environment, and then the authors describe an online algorithm for the early work maximization problem of competitive ratio $\frac{\sqrt{2m^2-2m+1}-1}{m-1}$, where m is the number of the machines. However, since the late work can be 0, no approximation or online algorithm is proposed for the late work objective.

☆ This work has been supported by the National Research, Development and Innovation Office – NKFIH, grant no. SNN 129178, and ED_18-2-2018-0006. The research of Péter Györgyi was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

* Corresponding author.
E-mail addresses: peter.gyorgyi@sztaki.hu (P. Györgyi), tamas.kis@sztaki.hu (T. Kis).

https://doi.org/10.1016/j.ejor.2020.03.032
0377-2217/

(ii) Sterna and Czerniachowska (2017) propose a polynomial time approximation scheme for the early work maximization problem with 2 machines, and it is not obvious how to get rid of some constant bound on the number of the machines. Further on, Chen et al. (2020b) describe a fully polynomial time approximation scheme for maximizing the early work on a fixed number of identical parallel machines.

(iii) We have observed that some variants of the resource leveling problem are equivalent to the early work maximization and the late work minimization problems. Briefly, the resource leveling problems we are referring to consist of a parallel machine environment and one more renewable resource required by a set of unit time jobs having a common deadline, and one aims at to minimize (maximize) the total resource usage above (below) a threshold. We are not aware of any published approximation algorithms for resource leveling problems in a parallel machine environment, but the results for the early- and late work problems can be transferred to this important subclass.

In this paper we propose a polynomial time approximation scheme for the early work maximization problem in an identical parallel machine environment, which we extend to the late work minimization problem in the same processing environment. By applying a concept of strong equivalence, we obtain analogous results for the maximization as well as the minimization variant of the resource leveling with unit time jobs problem on identical parallel machines. We emphasize that the number of identical parallel machines is part of the input for all problems studied, and the processing times of the jobs are arbitrary positive integer numbers in the early work maximization, and the late work minimization problems, while we have unit time jobs and arbitrary resource requirements in the resource leveling problems.

The results of this paper are theoretical in nature, the proposed algorithms are not intended for practical use. However, they provide new insight that can lead to efficient algorithms, and the technique developed, outlined in the last section, may be used for deriving approximation algorithms for other problems as well.

In Section 2 we precisely define the scheduling problems studied in this paper, and provide the necessary terminology. In Section 3 we summarize related work from the literature. In Section 4 we prove the equivalence of the late work minimization problem with the minimization variant of the resource leveling with unit time jobs problem, and an analogous result for the early work maximization problem and the maximization variant of the resource leveling problem. An inapproximability result is stated and proved for the late work minimization problem in Section 5. In Section 6 we describe a polynomial time approximation scheme for the early work maximization problem extended with machine capacity constraints, and in Section 7 we adapt the results of Section 6 to the late work minimization problem after shifting the objective function by a problem-data dependent value. By the results of Section 4, we obtain polynomial time approximation schemes for the two variants of the resource leveling problem as well. We conclude the paper in Section 8.

2. Problem formulation and terminology

In the late work minimization problem in a parallel machine environment, there is a set $\mathcal{J}$ of n jobs that have to be scheduled on m identical parallel machines. If it is not noted otherwise, the number of the machines is part of the input. Each job $j \in \mathcal{J}$ has a processing time p_j and there is a common due date d. The late work objective Y is to minimize the total amount of work scheduled after d, see Chen et al. (2016). That is, a schedule S specifies a machine $\mu_j(S) \in \{1, \dots, m\}$ and a starting time $t_j(S) \geq 0$ for each job. S is *feasible* if for each pair of distinct jobs j and k such that $\mu_j(S) = \mu_k(S)$, either $t_j(S) + p_j \leq t_k(S)$ or $t_k(S) + p_k \leq t_j(S)$. Throughout the paper we assume that there are no idle times between the jobs on any machine. The *late work* of a schedule S is $Y(S) = \sum_{i=1}^{m} \max\{0, \sum_{j \in J_i(S)} p_j - d\}$, where $J_i(S) = \{j \in \mathcal{J} \mid \mu_j(S) = i\}$. Later we will frequently refer to the sum of the job processing times $p_{sum} := \sum_{j \in \mathcal{J}} p_j$.

We add a further constraint to this problem. We introduce a bound N on the number of the jobs that can be scheduled on any of the machines. This is called *machine capacity*, see e.g. Woeginger (2005). Throughout the paper we assume that $m \cdot N \geq n$, otherwise there is no feasible solution for the problem. Note that machine capacity is not a common constraint for the late work minimization problem, but it will be useful later. However, by setting $N = n$, the capacity constraints become void, and we get back the familiar late work minimization problem.

Since the late work objective can be 0, and deciding whether a feasible schedule of zero late work exists or not is a strongly NP-hard decision problem (Chen et al., 2016), no approximation algorithm exists for this objective. However, by applying a standard trick, we can ensure that the objective function value is always positive, and approximating it becomes possible. We introduce a problem instance-dependent positive number T, and when approximating the optimum late work, we will consider the objective function $T + Y$.

There is another way to modify the objective function so that it allows us to achieve approximation results. The early work objective X, introduced by Błażewicz, Pesch, Sterna, and Werner (2005), which measures the total amount of work scheduled on the machines before d, is closely related to Y by the equation

$$X(S) = p_{sum} - Y(S) \quad \text{for any feasible schedule } S. \tag{1}$$

In the *resource leveling problem*, we have n jobs with unit processing times to be scheduled on m identical parallel machines in the time interval $[0, C]$, where C is a common deadline of all the jobs. Additionally, there is a renewable resource along with a soft limit L for the resource usage. Each job j has some requirement $a_j \geq 0$ from the resource. All problem data is integral. A *schedule* S specifies a machine $\mu_j(S) \in \{1, \dots, m\}$ and *starting time* $t_j(S) \in \{0, \dots, C-1\}$ for each job j. Without loss of generality, $m \cdot C \geq n$, otherwise no feasible schedule exists. Throughout the paper we assume that in any schedule, if $k < m$ jobs start at some time point t, then they occupy the first k machines. The goal is to find a feasible schedule S, where each job starts in $[0, C-1]$, and the total resource usage above L is minimized, i.e., we have to minimize $\tilde{Y}(S) := \sum_{t=0}^{C-1} \max\{0, \sum_{j \in J_t(S)} a_j - L\}$, where $J_t(S) = \{j \in \mathcal{J} \mid t_j(S) = t\}$, and $\sum_{j \in J_t(S)} a_j$ is the *total resource usage* of those jobs starting at time point t. A closely related problem is the maximization of the total resource usage below L over the scheduling horizon $[0, C]$, i.e., maximize $\tilde{X}(S) := \sum_{t=0}^{C-1} \min\{L, \sum_{j \in J_t(S)} a_j\}$. Let $a_{sum} := \sum_{j \in \mathcal{J}} a_j$. The two objective functions are related by the equation

$$\tilde{X}(S) = a_{sum} - \tilde{Y}(S) \quad \text{for any feasible schedule } S. \tag{2}$$

Notice the similarity of (1) and (2). As we will see, this is not a coincidence. Furthermore, since checking whether a feasible schedule S with $\tilde{Y}(S) = 0$ exists is a strongly NP-hard decision problem (Neumann & Zimmermann, 2000), for approximating the optimal solution we will use the objective function $\tilde{T} + \tilde{Y}$, where $\tilde{T}$ is an instance-dependent positive number. If $m \geq n$, then we get the project scheduling version of the resource leveling problem, i.e., there are no machines and arbitrary number of jobs can be started at the same time.

This paper uses the $\alpha|\beta|\gamma$ notation of Graham, Lawler, Lenstra, and Rinnooy Kan (1979), where α denotes the machine environment, β the additional constraints, and γ the objective function. In the α field we use P for arbitrary number of parallel machines

and $P2$ in case of two machines. In the β field, $d_j = d$ indicates that the jobs have a common due date, while $n_i \leq N$ indicates the capacity constraints of the machines. The symbols X and Y in the γ field refer to the early work, and to the late work criterion, respectively, and we use the symbols $\tilde{X}$ and $\tilde{Y}$ to denote the total resource usage below and above the limit L, respectively, in case of the resource leveling problem.

In this paper we describe approximation algorithms for the above mentioned, and some other combinatorial optimization problems. Our terminology closely follows that of Garey and Johnson (1979). A *minimization* (resp. *maximization*) problem Π is given by a set of instances $\mathcal{I}$, and each instance $I \in \mathcal{I}$ has a set of solutions $\mathcal{S}^I$, and an objective function $c^I : \mathcal{S}^I \to \mathbb{Q}$. Given any instance I, the goal is to find a feasible solution $s^* \in \mathcal{S}^I$ such that $c^I(s^*) = \min\{c^I(s) \mid s \in \mathcal{S}^I\}$ ($c^I(s^*) = \max\{c^I(s) \mid s \in \mathcal{S}^I\}$). Let $OPT(I)$ denote the optimum objective function value of problem instance I. A *factor ρ approximation algorithm* for a minimization (maximization) problem Π is a polynomial time algorithm A such that the objective function value, denoted by $A(I)$, of the solution found by the algorithm A on any problem instance $I \in \mathcal{I}$ satisfies $A(I) \leq \rho \cdot OPT(I)$ ($A(I) \geq \rho \cdot OPT(I)$). Naturally, $\rho \geq 1$ for minimization problems, and $0 < \rho \leq 1$ for maximization problems. Furthermore, a *polynomial time approximation scheme (PTAS)* for Π is a family of algorithms $\{A_\varepsilon\}_{\varepsilon > 0}$ such that A_ε is a factor $1 + \varepsilon$ approximation algorithm for Π if it is a minimization problem, or a factor $1 - \varepsilon$ approximation algorithm ($0 < \varepsilon < 1$), for Π if it is a maximization problem. In addition, a *fully polynomial time approximation scheme (FPTAS)* is like a PTAS, but the time complexity of each A_ε must be polynomial in $1/\varepsilon$ as well.

Let Π_1 and Π_2 be two optimization problems. We say that they are *strongly equivalent* if there exist bijective functions f and g, where f establishes a one-to-one correspondence between the instances of Π_1 and that of Π_2, whereas g establishes a one-to-one correspondence between the set of solutions of each instance I of Π_1 and that of $f(I)$ of Π_2 such that for each $S \in \mathcal{S}^I$, $c^I(S) = c^{f(I)}(g(S))$.

3. Previous work

In this section first we overview existing complexity and approximability results for scheduling problems with the total late work minimization, and the total early work maximization objective functions, but we abandon exact and heuristic methods as they are not directly related to our work. Then we briefly overview what is known about resource leveling in a parallel machine environment.

The total late work objective function (*late work* for short) is proposed by Błażewicz (1984), where the complexity of minimizing the total late work in a parallel machine environment is investigated. For non-preemptive jobs it is mentioned that minimizing the late work is NP-hard, while for preemptive jobs, a polynomial-time algorithm, based on network flows, is described. This approach is extended to uniform machines as well. Subsequently, several papers have appeared discussing the late work minimization problem in various processing environments. For the single machine environment, Potts and Van Wassenhove (1992b) describe an $O(n \log n)$ time algorithm for the problem with preemptive jobs, where each job has its own due date. Furthermore, the non-preemptive variant is shown to be NP-hard, and among other results, a pseudo-polynomial time algorithm is proposed for finding optimal solutions. Potts and Van Wassenhove (1992a) devise a fully polynomial time approximation scheme for the single machine non-preemptive late work minimization problem, which is extended to the total weighted late work problem by Kovalyov, Potts, and Van Wassenhove (1994), where the late work of each job is weighted by a job-specific positive number. For a two-machine flow shop, Błażewicz et al. (2005) prove that the late work minimization problem is NP-hard even if all the jobs have a common due date, and they also describe a dynamic programming based exact algorithm. A more complicated dynamic program is proposed for the two-machine job shop problem with the late work criterion by Błażewicz, Pesch, Sterna, and Werner (2007). Late work minimization in an open shop environment, with preemptive or with non-preemptive jobs, is studied in Błażewicz, Pesch, Sterna, and Werner (2004), where a number of complexity results are proved. For the parallel machine environment, Chen et al. (2016) prove that deciding whether a schedule with 0 late work exists is a strongly NP-hard decision problem, while if the number of machines is only 2, then it is binary NP-hard even if the jobs have a common due date. Furthermore, they describe an online algorithm for maximizing the early work of jobs that have to be scheduled in a given order. For several other complexity results not mentioned here, we refer to Sterna (2000, 2006, 2011).

A related problem is the minimization of the total tardiness on identical parallel machines, when the jobs have a common due date d. Kovalyov and Werner (2002) observe that without modifying the objective function, there is no hope for any approximation algorithm, like in the case of minimizing the total late work. Hence, they augment the objective function value by a positive constant b, and prove that the problem does not admit a factor $(1 + \varepsilon)$ approximation algorithm for any $0 < \varepsilon < 1/b$ unless P = NP. It follows that in order to have an (F)PTAS, b must depend polynomially on d or the job processing times. They also describe a fully polynomial time approximation scheme if $b = d$, and the number of the machines is fixed.

As for the early work, besides the paper of Chen et al. (2016), we mention Sterna and Czerniachowska (2017), where a PTAS is proposed for maximizing the early work in a parallel machine environment with 2 machines, where all the jobs have a common due date. Chen et al. (2020b) describe a fully polynomial time approximation scheme if the number of identical parallel machine is fixed. They also provide computation results for the previous PTAS as well as for the FPTAS on problem instances with 2 and 3 machines and up to 65 and 13 jobs, respectively.

Resource leveling is a well studied area of project scheduling, where a number of exact and heuristic methods are proposed for solving it for various objective functions and under various assumptions, see e.g., Kis (2005), Neumann and Zimmermann (2000), Rieck, Zimmermann, and Gather (2012), Verbeeck, Van Peteghem, Vanhoucke, Vansteenwegen, and Aghezzaf (2017). Drótos and Kis (2011) consider a dedicated parallel machine environment, and propose and exact method for solving resource leveling problems optimally with hundreds of jobs. In the same paper, some new complexity results are obtained.

Chen, Kovalev, Sterna, and Błażewicz (2020a) introduce the notion of mirror scheduling problems, which is a kind of strong equivalence. Two scheduling problems, Π_1 and Π_2, constitute a pair of *mirror scheduling problems* if there is a bijective mapping between their instances, and any solution S_1 of any instance I_1 of Π_1 can be mapped to a solution S_2 of the corresponding instance I_2 of Π_2 such that the objective function values of the two schedules are equal, and there is a mirror time point T and if a machine processes job j at time t in S_1, then the same machine processes j at time $T - t$ in S_2.

4. Equivalence of the late work minimization problem and the resource leveling problem

In this section we prove the equivalence of the late work minimization problem and the resource leveling problem in the sense defined at the end of Section 2.

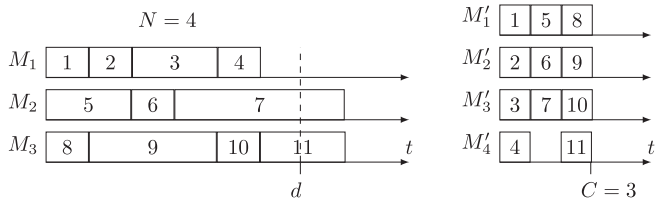

Fig. 1. Corresponding schedules for late work minimization problem and resource leveling problem.

Theorem 1. *The late work minimization problem $P|d_j = d, n_i \le N|Y$, and the resource leveling problem $P|p_j = 1|\tilde{Y}$ are equivalent.*

Proof. The proof consists of two parts. First, we define a bijective function between the set of instances of the late work minimization problem and the set of instances of the resource leveling problem with unit time jobs. Then, we consider an arbitrary pair of instances of the two problems (the pair is determined by the previous function) and we define another bijective function between the schedules of the two instances.

Consider an arbitrary instance I of the late work minimization problem (m machines, n jobs with processing times p_j ($j \in \{1, \dots, n\}$) and common due date d, and upper bound N on the number of jobs on each machine). The corresponding instance of the resource leveling problem has N machines, n jobs with processing times 1, resource requirements $a_j := p_j$ ($j = 1, \dots, n$), common deadline $C := m$, and resource limit $L := d$. Now we verify that the given mapping between the instances of the two problems is a bijection. Indeed, the function is injective (different instances of the late work minimization problem are mapped to different instances of the resource leveling problem), and surjective (for every instance I' of the resource leveling problem there is an instance I of the late work minimization problem such that I is mapped to I'), thus it is bijective.

Now, we describe a mapping from the set of feasible schedules of any instance of the late work minimization problem to that of the corresponding instance of the resource leveling problem. Let instance I of the late work minimization problem be fixed and let I' be the corresponding instance of resource leveling problem. Let S be any feasible schedule for the instance I, our function defines a schedule S' for I' based on S as follows. If a job j is the ℓth job scheduled on machine i in S then schedule the corresponding job of I' on machine ℓ at time $t_j(S') := i - 1$, for an illustration, see Fig. 1. □

The following series of claims will prove the theorem:

Claim 1. *S' is feasible for I'.*

Proof. Since there are at most N jobs scheduled on a machine in S, thus we assign each job to one of the N machines of I'. Furthermore, each job in I' has a unit processing time, hence the jobs do not overlap. □

Claim 2. *The mapping between the schedules for I and that for I' is a bijection.*

Proof. It is easy to see that the given mapping of schedules is injective. Moreover, let S' be any schedule for I'. We define S for I such that S is mapped to S' as follows. Suppose job j starts on M'_ℓ at time point $i - 1$ for some $i \in \{1, \dots, C\}$ in S', then j is the ℓ^{th} job on $\mu_j(S) = i$. Since in S', there is no idle machine among $M'_1, \dots, M'_\ell$ by definition, S is feasible, and the value of $t_j(S)$ is well defined. □

Claim 3. *If the late work of some schedule S for instance I is Y, then the objective function value of the corresponding schedule S' for I' is also Y.*

Proof. Consider the i^{th} machine M_i ($i \in \{1, \dots, m\}$) in S, let $\mathcal{J}_i$ denote the set of jobs scheduled on M_i in S. The late work on M_i is $\max\{0, \sum_{j\in\mathcal{J}_i} p_j - d\}$, thus $Y = \sum_{i=1}^{m} \max\{0, \sum_{j\in\mathcal{J}_i} p_j - d\}$. On the other hand, observe that the jobs of $\mathcal{J}_i$ are mapped to those jobs of the resource leveling problem that start at time point $i - 1$ in S'. The total resource requirement of these jobs exceeds L by $\max\{0, \sum_{j\in\mathcal{J}_i} a_j - L\}$, thus the objective function value of S' is $\sum_{i=1}^{C} \max\{0, \sum_{j\in\mathcal{J}_i} a_j - L\} = \sum_{i=1}^{m} \max\{0, \sum_{j\in\mathcal{J}_i} p_j - d\} = Y$, since $L = d$, $C = m$, and $p_j = a_j$ by the mapping defined above.

The above claims prove the theorem. □

By (1) and (2), we have the following:

Corollary 1. *The early work maximization problem $P|d_j = d, n_i \le N|X$, and the resource leveling problem $P|p_j = 1|\tilde{X}$ are equivalent.*

5. Inapproximability of $P2|d_j = d|c' + Y$

In this section we prove that if we simply add a value c' to Y in the objective function of the late work minimization problem, where c' is a fixed positive number, then it is impossible to get an approximation algorithm of factor smaller than $\frac{c'+1}{c'}$ unless $P = NP$. We will use the following result of Chen et al. (2016):

Theorem 2 (Theorem 2 in Chen et al., 2016)**.** *The problem $P2|d_j = d|Y$ is NP-hard. In particular, it is NP-hard to decide if a feasible schedule of total late work 0 exists.*

The following statement and its proof is analogous to that of Theorem 2 of Kovalyov and Werner (2002) for the inapproximability of $Pm|d_j = d|b + \sum T_j$.[1]

Proposition 1. *Let c' be a positive constant. Then for any $0 < \varepsilon < 1/c'$, there is no $(1 + \varepsilon)$-approximation algorithm for $P2|d_j = d|c' + Y$ unless $P = NP$.*

Proof. Suppose we have a factor $1 + \varepsilon$ approximation algorithm for $P2|d_j = d|c' + Y$ for some $0 < \varepsilon < 1/c'$. We show how to apply this approximation algorithm to decide if for any instance of $P2|d_j = d|Y$ a feasible schedule of total late work 0 exists. However, the latter decision problem is NP-hard by Theorem 2, which implies our claim.

Consider any instance of $P2|d_j = d|c' + Y$. If the approximation algorithm returns a solution of value c', then clearly, there is a schedule of 0 late work. Now suppose the approximation algorithm returns a solution of value at least $c' + 1$ (no value between c' and $c' + 1$ is possible, because all problem data is integral). Indirectly, assume that there is a schedule of total late work 0, and hence, the optimum solution value is c'. But then $c' + 1 \le (1 + \varepsilon)c' < c' + 1$ must hold, where the first inequality follows from the approximation factor and the second form $\varepsilon < 1/c'$. This is a contradiction, thus all feasible schedules must have total late work at least 1. □

6. A PTAS for $P|d_j = d, n_i \le N|X$

In this section we describe a PTAS for $P|d_j = d, n_i \le N|X$. Note that the machine capacity N is a positive integer such that $m \cdot N \ge n$, where n is the number of the jobs, and m is the number of identical parallel machines.

We will devise two algorithms (both parameterized by ε), and we will run both of them on the same input, and finally, we will choose the better of the two schedules obtained as the output of the algorithm. The first family of algorithms, described in Section 6.1, has an approximation factor of $(1 - 4\varepsilon)$ if the optimum value is at least $\varepsilon \cdot m \cdot d$. In contrast, the approximation algorithm

[1] We thank a referee for calling our attention to this paper.

presented in Section 6.2 is of factor $1-2\varepsilon$ if the optimum value is smaller than $\varepsilon \cdot m \cdot d$. Running both methods on the same input guarantees an approximation factor of $(1-4\varepsilon)$.

After some preliminary observations, we will describe the two algorithms along with the proofs of their soundness, and in the end we combine them to obtain the PTAS.

Throughout this section, S^* denotes an optimal schedule for an instance of $P|d_j = d, n_i \le N|X$.

6.1. Family of algorithms for the case $X(S^*) \ge \varepsilon \cdot m \cdot d$

In this section we describe a family of algorithms $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$, such that $\mathcal{A}_\varepsilon$ is a factor $(1-4\varepsilon)$ approximation algorithm for the problem $P|d_j = d, n_i \le N|X$ under the condition $X(S^*) \ge \varepsilon \cdot m \cdot d$.

We start by observing that if a job starts after d then we do not have to deal with its exact starting time and with its machine assignment, because the total processing time of this job is late work. We can schedule these jobs from any time point after d on any machine where we do not violate the machine capacity constraints.

Let $\varepsilon > 0$ be fixed. We divide the set of jobs into three subsets, huge, big and small. The set of *huge jobs* is $\mathcal{H} := \{j \in \mathcal{J} \mid p_j \ge d\}$, the set of *big jobs* is $\mathcal{B} := \{j \in \mathcal{J} \mid \varepsilon^2\, d \le p_j < d\}$, and the remaining jobs are *small*.

Proposition 2. *If there are at least m huge jobs, then scheduling m, arbitrarily chosen huge jobs on m distinct machines, and the rest of the jobs arbitrarily, yields an optimal schedule both for the maximum early work and the minimum late work objectives.*

Proof. Let S' be the schedule constructed as described in the statement of the proposition. Then $X(S') = m \cdot d$, which is the maximum possible early work. By Eq. (1), S' has minimum late work as well, thus it is optimal for both objective functions. □

Proposition 3. *If $|\mathcal{H}| \le m-1$, then there exists an optimal schedule for the maximum early work as well as for the minimum late work objectives such that the huge jobs are scheduled on $|\mathcal{H}|$ distinct machines.*

Proof. Let S^* be an optimal schedule for the early work (as well as for the late work) objective with the maximum number of machines on which a huge job is scheduled. Indirectly, suppose less than $|\mathcal{H}|$ machines process at least one huge job, hence, there exists a machine M_1 processing at least two huge jobs, say j_1 and j_2, in this order. Since there are at most $m-1$ huge jobs, there exists a machine M^* (in fact there are at least two), which does not process any huge jobs. If less than N jobs are scheduled on M^*, then move job j_2 from M_1 to M^*, otherwise swap job j_2 with any of the jobs scheduled on M^*, and let S' be the resulting schedule. Clearly, the machine capacities are respected by S', and both of the machines M^* and M_1 work in the period $[0, d]$ in S', while the work assigned to any other machine is the same in both schedules. Hence, $X(S') \ge X(S^*)$. Therefore, S' is optimal for the early work objective, and by Eq. (1), for the late work objective as well. However, in S' more machines process at least one huge job than in S^*, a contradiction. □

From now on, we assume that there are at most $m-1$ huge jobs, and we fix an optimal schedule S^* in which the huge jobs are scheduled on distinct machines.

Our algorithm has three main phases: first, we schedule all of the huge jobs, and some of the big jobs such that they get a starting time smaller than d, then we schedule some of the small jobs such that they get a starting time smaller than d, and finally, we schedule the remaining big and small jobs, if any, arbitrarily while respecting the machine capacity constraints.

For each big job j we round down its processing time p_j to the greatest integer $p'_j := \lceil \varepsilon^2\ d(1+\varepsilon)^k \rceil$ by selecting $k \in \mathbb{Z}$ such that $p'_j \le p_j$. Since we have $\varepsilon^2 d \le p_j < d$ for each big job j, the number of the different p'_j values is bounded by the constant $k_1 := \lfloor \log_{1+\varepsilon}(1/\varepsilon^2) \rfloor + 1$ that depends on the fixed ε only. Let $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{k_1}$ denote the sets of the big jobs with the same rounded processing times, i.e., $\mathcal{B}_h := \{j \in \mathcal{J} : p'_j = \lceil (1+\varepsilon)^{h-1} \cdot \varepsilon^2\ d \rceil\}$ ($\mathcal{B}_h = \emptyset$ is possible).

For each machine without a huge job, we guess the number of the big jobs from each set $\mathcal{B}_h$ that start before d. This guess can be described by an *assignment* A, which consists of k_1 numbers $(\gamma_1, \gamma_2, \ldots, \gamma_{k_1})$, where γ_h describes the number of the jobs from $\mathcal{B}_h$. A big job assignment $(\gamma_1, \gamma_2, \ldots, \gamma_{k_1})$ is *feasible*, if it does not violate the constraint on the number of the jobs on a machine, i.e., $\sum_{h=1}^{k_1} \gamma_h \le N$, and all the selected jobs can be started before d. To verify the latter condition, it suffices to schedule the selected jobs in any order such that the longest job is scheduled last, which ensures that the last job starts as early as possible. Let k_2 be the number of possible big job assignments. Since the total number of big jobs that may start before d on a machine is at most $\lfloor 1/\varepsilon^2 \rfloor$, we have $k_2 \le k_1^{\lfloor 1/\varepsilon^2 \rfloor}$. Let $A_1, A_2, \ldots, A_{k_2}$ denote the different feasible big job assignments.

A *layout* is a k_2 tuple $(t_1, t_2, \ldots, t_{k_2})$ that specifies for each feasible assignment the number of the machines that uses it. Let γ_{ih} denote the number of big jobs from $\mathcal{B}_h$ assigned by A_i. A layout is *feasible* if and only if $\sum_{i=1}^{k_2} t_i \gamma_{ih} \le |\mathcal{B}_h|$ for each $h = 1, \ldots, k_1$. The number of feasible tuples is bounded by the number of non-negative, integer solutions of the inequality $\sum_{i=1}^{k_2} t_i \le m - |\mathcal{H}|$, which is bounded by $\binom{m-|\mathcal{H}|+k_2}{k_2}$, a polynomial in the size of the input, since k_2 is a constant (that depends on ε only). In Algorithm A, we examine each big job layout and get a complete schedule for each of them.

Algorithm A

1. Determine the set of feasible layouts.
2. For each layout t, perform Steps 3–6.
3. Assign the huge jobs of $\mathcal{H}$ to machines $M_1, \ldots, M_{|\mathcal{H}|}$ arbitrarily, and big jobs to the remaining $m - |\mathcal{H}|$ machines according to t (t_i machines use assignment A_i)
4. On each machine, schedule the assigned jobs from time point 0 on in arbitrary order.
5. If $N \ge n$, then invoke Algorithm B, otherwise invoke Algorithm C to schedule small jobs.
6. Schedule the remaining jobs (small and big, if any) on the machines arbitrarily such that no machine receives more than N jobs in total (including the pre-assigned huge and big jobs).
7. Output S_A, which is the best schedule found in Steps 2–6.

Now we turn to Algorithms B and C for scheduling small jobs. Algorithm B is a simple greedy method which works only if there are no machine capacity constraints, i.e., $N \ge n$.

Algorithm B

Input: partial schedule of big jobs

1. For $i = 1, \ldots, m$ do:
2. Schedule a maximal subset of small jobs on machine M_i after the big jobs without idle time such that no small job finishes after d.

Observe that the above method may assign a lot of small jobs to a machine, thus it may not yield a feasible schedule if $N < n$.

Algorithm C is much more complicated. Let $\mathcal{J}^{small}$ denote the set of small jobs, $P_i^{small} \ge 0$ the idle time on machine i before d,

and n_i^{small} the number of the jobs that can be scheduled on machine i after the partial schedule of big jobs, i.e., n_i^{small} is the difference between N and the number of the big jobs assigned to machine M_i. Note that $P_i^{small} = 0$ if a huge job is assigned to machine M_i.

Our goal is to maximize the early work of the small jobs for a fixed assignment of big and huge jobs. To simplify our problem, we only want to maximize the total processing time of the small jobs that a machine completes before d. This may decrease the objective function value of the final schedule, but we will show that this error is negligible.

We can model the above problem with an integer program. We introduce $n \cdot (m+1)$ binary variables $x_{i,j}$ $(i = 0, 1, 2, \ldots, m,\ j = 1, 2, \ldots, n)$, where $x_{0,j} = 1$ means that we do not schedule job j to any machine before d, while in case of $1 \le i \le m$, $x_{i,j} = 1$ means that job j will be scheduled on machine i, and will be completed not later than d.

$$\max \sum_{i=1}^{m} \sum_{j \in \mathcal{J}^{small}} x_{i,j} p_j \tag{3}$$

$s.t.$

$$\sum_{j \in \mathcal{J}^{small}} x_{i,j} p_j \le P_i^{small}, \qquad i = 1, \ldots, m, \tag{4}$$

$$\sum_{j \in \mathcal{J}^{small}} x_{i,j} \le n_i^{small}, \qquad i = 1, \ldots, m, \tag{5}$$

$$\sum_{i=0}^{m} x_{i,j} = 1, \qquad j \in \mathcal{J}^{small}, \tag{6}$$

$$x_{i,j} \in \{0, 1\}, \qquad i = 0, \ldots, m,\ j \in \mathcal{J}^{small}. \tag{7}$$

We get the LP-relaxation of the above integer program by replacing $x_{i,j} \in \{0, 1\}$ with $x_{i,j} \ge 0$ in the constraints (7).

Algorithm C

Input: partial schedule of big jobs

1. Determine the values P_i^{small}, n_i^{small} for $i = 1, \ldots, m$.
2. Solve the LP-relaxation of (3)–(7), and let $\bar{x}$ be a basic optimal solution.
3. For $i = 1, \ldots, m$, if $\bar{x}_{i,j} = 1$ for a job j, then assign that job to machine i.
4. For each machine, schedule the assigned jobs right after the big jobs without idle times in arbitrary order.

Observe that fractional jobs of the optimal LP solution are not assigned to any machine by Algorithm C, but they will be scheduled by Step 6 of Algorithm A.

The proofs of the following two claims easily follow from the definitions.

Proposition 4. *S_A is feasible.*

Proposition 5. *The time complexity of Algorithm B is polynomially bounded in the size of the input.*

Proposition 6. *The time complexity of Algorithm C is polynomially bounded in the size of the input.*

Proof. We can determine a basic solution of a linear program with nm variables and $n + 2m$ constraints in two steps. First, apply a polynomial time interior-point algorithm to find a pair of primal-dual optimal solutions, and then, we can use Megiddo's method to determine a basic solution $\bar{x}$ for the primal program, see e.g., Wright (1997). The other steps of Algorithm C require linear time. □

Proposition 7. *The time complexity of Algorithm A is polynomially bounded in the size of the input.*

Proof. Recall that the number of the feasible layouts is polynomial (at most $\binom{m+k_2}{k_2}$). Each of the Steps 3–6 requires $O(nm)$ time, except Step 5 if it invokes Algorithm C, but it is also polynomial due to Proposition 6. □

Without loss of generality, we assume that in S^* the huge and big jobs precede the small jobs on each machine, and the big jobs are scheduled in non-decreasing processing time order on each machine. We introduce an intermediate schedule S_{int}: it is the same as S^* except that the processing time of each big job is rounded as in Algorithm A. That is, the processing time of each big job is rounded down to the greatest number of the form $\lceil \varepsilon^2\, d(1+\varepsilon)^k \rceil$, $(k \in \mathbb{Z})$, and after rounding we re-schedule the jobs on each machine in the same order as in S^*, but with the decreased processing times of the big jobs. By considering those big jobs on the machines that start before d in S_{int}, we can uniquely identify an assignment of big jobs for each machine. Therefore, we can determine the layout t^* of the big jobs that start before d in S_{int}. Now we state and prove the main result of this section.

Theorem 3. *If $X(S^*) \ge \varepsilon \cdot m \cdot d$, then Algorithm A is a factor $(1 - 4\varepsilon)$ approximation algorithm for $P|d_j = d, n_i \le N|X$.*

Proof. Recall that S_{int} is the schedule obtained from S^* by rounding down the processing time of each big job, and shifting the jobs to the left, if necessary, to eliminate any idle times (created by rounding) on the machines. Since $p_j/(1+\varepsilon) < p'_j \le p_j$, we have $X(S_{int}) \ge X(S^*)/(1+\varepsilon) \ge (1-\varepsilon)X(S^*)$. Let t^* be the layout of big jobs corresponding to S_{int}. Algorithm A will consider the layout t^* at some iteration, and let S be the schedule created from t^*. Since $X(S_A) \ge X(S)$, it suffices to prove that $X(S) \ge (1 - 4\varepsilon)X(S^*)$. To achieve this, we proceed by proving a series of lemmas. □

Lemma 1. *If $N \ge n$ and $X(S^*) \ge \varepsilon \cdot m \cdot d$, then $X(S) \ge (1-\varepsilon)X(S^*)$.*

Proof. If Algorithm B schedules all the small jobs when creating schedule S, then the only jobs finishing after d can be big and huge jobs. Since the set of big and huge jobs that start before d in schedule S contains all the big and huge jobs that start before d in schedule S_{int}, we get $X(S) \ge X(S_{int})$.

If there is at least one small job that remains unscheduled by Algorithm B, then consider the early work in S. We know that the total processing time on each machine is at least $d(1-\varepsilon^2)$ due the condition of Step 2 of Algorithm B. Hence, $X(S) \ge md(1-\varepsilon^2)$. Since $X(S) \le X(S^*) \le m \cdot d$, and $X(S^*) \ge \varepsilon \cdot m \cdot d$ by assumption, we derive

$$X(S) \ge (1-\varepsilon^2)d \cdot m \ge (1-\varepsilon)X(S^*),$$

as claimed. □

Proposition 8. *If $N < n$, then $X(S) \ge X(S_{int}) - 3\varepsilon^2 \cdot d \cdot m$.*

Proof. Consider Algorithm C, when it creates S. It solves (3)–(7) and $\bar{x}$ is the optimal basic solution that we get from the algorithm. Recall that if $i \ge 1$ then $\bar{x}_{i,j} = 1$ if and only if job j is assigned to machine i by Algorithm C. We introduce another integer solution x' of (3)–(7). Let $x'_{i,j} := 1$, if a small job j completes before d on machine i in S_{int}, otherwise, $x'_{i,j} := 0$. Note that x' is a feasible solution, because S_{int} is a feasible schedule.

Let $v(x)$ denote the objective function value of a solution x of (3)–(7), OPT_{IP} the optimum value of (3)–(7) and OPT_{LP} the optimum value of its linear relaxation. For any feasible solution x of (3)–(7), we have $OPT_{LP} \ge OPT_{IP} \ge v(x)$. Let X_{int}^{small} denote the early work of

the small jobs in S_{int} and X_S^{small} the same in S. Observe that $v(x')$, which is the total early work of the small jobs that complete before d in S_{int}, is at least $X_{int}^{small} - \varepsilon^2 dm$, because there is at most one small job on each machine that starts before, and ends after d, and recall that each small job is shorter than $\varepsilon^2 d$. Then

$$X_S^{small} \geq v(\lfloor \bar{x} \rfloor) \geq OPT_{LP} - 2\varepsilon^2 dm \geq OPT_{IP} - 2\varepsilon^2 dm \geq v(x') - 2\varepsilon^2 dm \geq X_{int}^{small} - 3\varepsilon^2 dm.$$

The first inequality is trivial, while we have already proved the last three inequalities. It remained to prove the second inequality, i.e., $v(\lfloor \bar{x} \rfloor) \geq OPT_{LP} - 2\varepsilon^2 dm$. Let e denote the number of the small jobs j with $\bar{x}_{i,j} = 1$ for some i ($i = 0, \ldots, m$) in Algorithm C, and $f := n - e$ the number of the 'fractionally assigned' small jobs. Note that for each of these small jobs, we have $i_1 \neq i_2$ ($0 \leq i_1, i_2 \leq m$) such that $\bar{x}_{i_1,j}, \bar{x}_{i_2,j} > 0$). Since $\bar{x}$ is a basic solution there are at most $n + 2m$ non-zero values among its coordinates. Hence, we have $e + 2f \leq n + 2m$, therefore, we have $f \leq 2m$. To sum up, we have

$$OPT_{LP} = \sum_{i=1}^{m} \left(\sum_{j:\bar{x}_{i,j}=1} p_j + \sum_{j \text{ frac. assigned}} \bar{x}_{i,j} p_j \right) = v(\lfloor \bar{x} \rfloor) + \sum_{j \text{ frac. assigned}} p_j \sum_{i=1}^{m} \bar{x}_{i,j} \leq v(\lfloor \bar{x} \rfloor) + 2\varepsilon^2 md,$$

where the last inequality follows from $f \leq 2m$, from $p_j \leq \varepsilon^2 d$ for each small job j, and from $\sum_{i=1}^{m} \bar{x}_{i,j} \leq 1$.

Finally, observe that $X_S^{small} \geq X_{int}^{small} - 3\varepsilon^2 dm$ implies $X(S) \geq X(S_{int}) - 3\varepsilon^2 dm$, since the set of big and huge jobs that start before d in S contains those of schedule S_{int}. □

Lemma 2. *If $N < n$ and $X(S^*) \geq \varepsilon \cdot m \cdot d$, then $X(S) \geq (1 - 4\varepsilon) X(S^*)$.*

Proof. By Proposition 8, $X(S) \geq X(S_{int}) - 3\varepsilon^2 \cdot d \cdot m$. Therefore, using the assumption of the lemma, we derive

$$X(S) \geq X(S_{int}) - 3\varepsilon^2 \cdot d \cdot m \geq X(S^*)(1 - \varepsilon) - 3\varepsilon X(S^*) = (1 - 4\varepsilon) X(S^*).$$

Now we can finish the proof of Theorem 3. We have proved that Algorithm A creates a feasible schedule S_A (Proposition 4) in polynomial time (Proposition 7) such that $X(S_A) \geq (1 - 4\varepsilon) X(S^*)$ (Lemmas 1–2), thus the theorem is proved. □

Theorem 3 has a strong assumption, namely, $X(S^*) \geq \varepsilon \cdot m \cdot d$. In the next section, we describe a complementary method, which works if $X(S^*) < \varepsilon \cdot m \cdot d$.

6.2. Approximation algorithm for the case $X(S^*) < \varepsilon \cdot m \cdot d$

We will show that if $X(S^*) < \varepsilon \cdot m \cdot d$, then scheduling the jobs in longest-processing-time-first order[2] by list-scheduling while respecting the capacity constraints of the machines yields an approximation algorithm both for minimizing the late work and for maximizing the early work as well. Recall the *list-scheduling* method of Graham (1969) for scheduling jobs on parallel machines. It processes the jobs in a given order, and it always schedules the next job on the least loaded machine. In order to take into account the capacity constraints of the machines, we will use the following variant of list-scheduling.

Algorithm LPT

Input: set of jobs, number of machines m, and common machine capacity N.

1. Let $n_i := 0$, and $L_i := 0$ for $i = 1, \ldots, m$.
2. Schedule the jobs in longest-processing-time-first order, ties are broken arbitrarily. When processing the next job j, choose the machine with minimum L_i value among those machines with $n_i < N$, and break ties arbitrarily. Let i be the index of the machine chosen. Then set $t_j(S_{LPT}) = L_i$, $\mu_j(S_{LPT}) := i$, $L_i := L_i + p_j$ and $n_i := n_i + 1$.
3. Return S_{LPT}.

The schedule S_{LPT} computed by the algorithm satisfies the following properties.

Theorem 4. *If $X(S^*) < \varepsilon \cdot m \cdot d$ and $\varepsilon \leq 1/3$, then $X(S_{LPT}) \geq (1 - 2\varepsilon) X(S^*)$ and $c \cdot p_{sum} + Y(S_{LPT}) \leq (1 + 2\varepsilon/c)(c \cdot p_{sum} + Y(S^*))$.*

Proof. First, we prove $X(S_{LPT}) \geq (1 - 2\varepsilon) X(S^*)$, and then we derive from it the second statement of the theorem. Since $X(S^*) \leq \varepsilon \cdot m \cdot d$, there can be at most $m - 1$ jobs of processing time at least εd. Since $X(S_{LPT}) \leq X(S^*)$, we can also deduce that in S_{LPT} there is a machine on which the total processing time of the jobs is less than εd.

First suppose that all jobs start *before* εd in S_{LPT}. Since there are $k \leq m - 1$ jobs of processing time at least εd, all these *long jobs* start on distinct machines in S_{LPT}, since these are the longest k jobs. All the remaining jobs have a processing time smaller than εd, and they are scheduled on the remaining $m - k$ machines. Therefore, the work finishes by time $2\varepsilon d$ on the remaining machines. Since $\varepsilon \leq 1/3$, the jobs, if any, that do not finish before d in S_{LPT} must be long jobs. Since the long jobs are scheduled on distinct machines in S_{LPT}, there is no way to decrease the late work of this schedule, or equivalently, to increase the early work, thus, S_{LPT} must be optimal for both objectives.

Now suppose there is a job j which starts at or after εd in S_{LPT}. Then there is a machine M^* in S_{LPT} with N jobs and the total processing time of these jobs is smaller than εd, otherwise either job j could be scheduled on M^* (which would contradict the rules of the list-scheduling algorithm), or $X(S_{LPT}) \geq \varepsilon \cdot m \cdot d$ (which would contradict the assumption $X(S^*) < \varepsilon \cdot m \cdot d$, since S_{LPT} is a feasible schedule, and S^* is an optimal schedule, thus $\varepsilon \cdot m \cdot d \leq X(S_{LPT}) \leq X(S^*)$).

We claim that on any machine, the total processing time of those jobs that start at or after εd is at most εd. This is so, because the jobs are scheduled in non-increasing processing time order, and no machine may receive more than N jobs. Consequently, if a job is started at or later than εd on some machine, it has a processing time not greater than the shortest processing time on M^*. Hence, the total processing time of the jobs scheduled on M^* is indeed an upper bound on the total processing time of those jobs started at or later than εd on any single machine.

By our claim, if there are only short jobs (of processing time smaller than εd) on a machine, then the total work assigned to it by S_{LPT} is at most $3\varepsilon d$. Hence, all these jobs finish by d, since $\varepsilon \leq 1/3$. Consequently, if a job finishes after d in S_{LPT}, then it must be scheduled on a machine with a long job. Let g be the number of those machines on which some job is late, i.e., finishes after d in S_{LPT}. Consider any of these g machines. It has a long job scheduled first, and then some short jobs. The total processing time of these short jobs is at most εd, since each of them starts after εd. Hence, the late work can be decreased by at most $g \cdot \varepsilon d$ by scheduling some of the short jobs early in a more clever way than in S_{LPT}. Consequently, $X(S_{LPT}) + g \cdot \varepsilon d \geq X(S^*)$.

Now, we bound gd. As we have observed, if a machine has some late work on it in S_{LPT}, then it has a long job, and some short jobs of total processing time at most εd. Hence, the length of the long job must be at least $d(1 - \varepsilon)$. Therefore, $X(S^*) \geq gd(1 - \varepsilon)$. Using this observation, we obtain the first statement:

$$X(S_{LPT}) \geq X(S^*) - \varepsilon \cdot gd \geq X(S^*) - \varepsilon X(S^*)/(1 - \varepsilon) \geq X(S^*)(1 - 2\varepsilon),$$

where the last inequality follows from $\varepsilon/(1 - \varepsilon) \leq 2\varepsilon$ if $0 < \varepsilon \leq 1/2$.

[2] the jobs are scheduled in non-increasing processing time order

Now we derive the second statement of the theorem. By Eq. (1), $Y(S_{LPT}) = p_{sum} - X(S_{LPT})$. Hence, we compute

$$\begin{aligned} Y(S_{LPT}) &= p_{sum} - X(S_{LPT}) \le p_{sum} - X(S^*)(1-2\varepsilon) \\ &= p_{sum} - (p_{sum} - Y(S^*))(1-2\varepsilon) \\ &= p_{sum} - (p_{sum} - 2\varepsilon p_{sum} - Y(S^*) + 2\varepsilon Y(S^*)) \\ &\le Y(S^*) + 2\varepsilon p_{sum}. \end{aligned}$$

To finish the proof, observe that

$$\begin{aligned} c \cdot p_{sum} + Y(S_{LPT}) &\le c \cdot p_{sum} + Y(S^*) + 2\varepsilon p_{sum} \\ &\le (1 + 2\varepsilon/c)(c \cdot p_{sum} + Y(S^*)). \end{aligned}$$

□

6.3. The combined method

In this section we combine the methods of Sections 6.1 and 6.2 to get a PTAS for $P|d_j = d, n_i \le N|X$.

Theorem 5. *There is a PTAS for $P|d_j = d, n_i \le N|X$.*

Proof. By Theorems 3 and 4, the following algorithm is a PTAS for $P|d_j = d, n_i \le N|X$.

Algorithm PTAS

Input: problem instance and parameter $0 < \varepsilon \le 1/3$.

1. Run Algorithm A and let S_A be the best schedule found.
2. Run Algorithm *LPT*, and let S_{LPT} be the schedule obtained.
3. If $X(S_A) \ge X(S_{LPT})$, then output S_A, else output S_{LPT}.

Since the conditions of Theorems 3 and 4 are complementary, it follows that Algorithm *PTAS* always outputs a solution of value at least $(1-4\varepsilon)$ times the optimum. The time complexity in either case is polynomial in the size of the input, hence, the algorithm is indeed a PTAS for our scheduling problem.

The time complexity of the combined method is dominated by that of Algorithm A, which is polynomial in the size of the input by Proposition 7, but exponential in $1/\varepsilon$. □

Since our result is valid even if $N \ge n$, we have the following corollary:

Corollary 2. *There is a PTAS for $P|d_j = d|X$.*

By Corollary 1, we immediately get an analogous result for the maximization variant of resource leveling problem:

Corollary 3. *There is a PTAS for the resource leveling problem $P|p_j = 1|\tilde{X}$.*

7. A PTAS for $P|d_j = d, n_i \le N|c \cdot p_{sum} + Y$

In this section we adapt the PTAS of Section 6 to the problem $P|d_j = d, n_i \le N|c \cdot p_{sum} + Y$. Throughout this section, S^* denotes an optimal solution of a problem instance for the late work objective, and by Eq. (1) for the early work objective as well.

7.1. The first family of algorithms

In this section we describe a family of algorithms $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$, such that $\mathcal{A}_\varepsilon$ is a factor $(1 + c_0 \cdot \varepsilon)$ approximation algorithm for the problem $P|d_j = d, n_i \le N|c \cdot p_{sum} + Y$ under the condition $X(S^*) \ge \varepsilon \cdot m \cdot d$, where c_0 is a universal constant, independent of ε and the problem instances.

Recall the definition of huge, big and small jobs from Section 6, we use the same partitioning of the set of jobs in this section as well.

By Propositions 2 and 3, it suffices to consider the case when there are at most $m-1$ huge jobs. However, in this section we round up the processing time p_j of each big job j to the smallest integer of the form $\lfloor \varepsilon^2 d(1+\varepsilon)^k \rfloor$, where $k \in \mathbb{Z}_{\ge 0}$. Since $\varepsilon^2 d \le p_j < d$ for each big job, there are at most $k_1 := \lfloor \log_{1+\varepsilon} 1/\varepsilon^2 \rfloor + 1$ distinct rounded processing times of the big jobs. Let $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{k_1}$ denote the sets of the big jobs with the same rounded processing times, i.e., $\mathcal{B}_h := \{j \in \mathcal{J} : p'_j = \lfloor \varepsilon^2 \, d \cdot (1+\varepsilon)^{h-1} \rfloor\}$ ($\mathcal{B}_h = \emptyset$ is possible). We also define the assignments of big jobs to machines and the layouts in the same way as in Section 6, but using the jobs classes $\mathcal{B}_h$ just defined.

Theorem 6. *If $X(S^*) \ge \varepsilon \cdot m \cdot d$, then Algorithm A is a factor $(1 + 4\varepsilon/c)$ approximation algorithm for $P|d_j = d, n_i \le N|c \cdot p_{sum} + Y$.*

Proof. Let S_{int} be the schedule obtained from S^* by rounding up the processing time of each big job, and shifting the jobs to the right, if necessary, so that the jobs do not overlap on any machine. Let t^* be the layout of big jobs corresponding to S_{int} (defined as in Section 6). Algorithm A will consider the layout t^* at some iteration, and let S be the schedule created from t^*. Since $Y(S_A) \le Y(S)$, it suffices to prove that $c \cdot p_{sum} + Y(S) \le (1 + O(\varepsilon))(c \cdot p_{sum} + Y(S^*))$, and this is what we accomplish subsequently. The claimed approximation factor is proved by a series of three lemmas. □

Lemma 3. $c \cdot p_{sum} + Y(S_{int}) \le (1 + \varepsilon/c)(c \cdot p_{sum} + Y(S^*))$.

Proof. Observe that the rounding procedure increases the late work by at most εp_{sum} (recall that $p_{sum} := \sum_{j \in \mathcal{J}} p_j$). Hence, we have

$$c \cdot p_{sum} + Y(S_{int}) \le c \cdot p_{sum} + Y(S^*) + \varepsilon p_{sum} \le (1 + \varepsilon/c)(c \cdot p_{sum} + Y(S^*).$$

□

Lemma 4. *If $N \ge n$ and $X(S^*) \ge \varepsilon \cdot m \cdot d$, then $c \cdot p_{sum} + Y(S) \le (1 + 2\varepsilon/c)(c \cdot p_{sum} + Y(S^*))$.*

Proof. If Algorithm B schedules all the small jobs when creating schedule S, then the only jobs finishing after d can be big and huge jobs. Since the set of big and huge jobs that start before d in schedule S contains all the big and huge jobs that start before d in schedule S_{int}, we get $Y(S) \le Y(S_{int})$.

If there is at least one small job that remains unscheduled after Step 5 of Algorithm A, then consider the early work in S. We know that the total processing time on each machine is at least $(1-\varepsilon^2)d$ due to the condition in Step 2 of Algorithm B, thus $X(S) \ge (1-\varepsilon^2)d \cdot m$. On the other hand, $X(S_{int}) \le d \cdot m$ is trivial, thus we have $Y(S) \le Y(S_{int}) + \varepsilon^2 d \cdot m$ due to (1). Finally, we have

$$\begin{aligned} c \cdot p_{sum} + Y(S) &\le c \cdot p_{sum} + Y(S_{int}) + \varepsilon^2 d \cdot m \\ &\le c \cdot p_{sum} + Y(S_{int}) + \varepsilon X(S^*) \\ &\le (1 + \varepsilon/c)(c \cdot p_{sum} + Y(S^*)) + \varepsilon(p_{sum} - Y(S^*)) \\ &\le (1 + 2\varepsilon/c)(c \cdot p_{sum} + Y(S^*)), \end{aligned}$$

where the second inequality follows from the assumption $X(S^*) \ge \varepsilon \cdot m \cdot d$, and the third from Lemma 3 and Eq. (1). □

Lemma 5. *If $N < n$ and $X(S^*) \ge \varepsilon \cdot m \cdot d$, then $c \cdot p_{sum} + Y(S) \le (1 + 4\varepsilon/c)(c \cdot p_{sum} + Y(S^*))$.*

Proof. By Proposition 8 and Eq. (1), we have $Y(S) \le Y(S_{int}) + 3\varepsilon^2 dm$. Therefore,

$$\begin{aligned} c \cdot p_{sum} + Y(S) &\le c \cdot p_{sum} + Y(S_{int}) + 3\varepsilon^2 dm \\ &\le c \cdot p_{sum} + Y(S_{int}) + 3\varepsilon X(S^*) \\ &\le (1 + \varepsilon/c)(c \cdot p_{sum} + Y(S^*)) + 3\varepsilon(p_{sum} - Y(S^*)) \\ &\le (1 + 4\varepsilon/c)(c \cdot p_{sum} + Y(S^*)), \end{aligned}$$

where the second inequality follows from the assumption $X(S^*) \ge \varepsilon \cdot m \cdot d$, and the third from Lemma 3 and Eq. (1).

Now we can finish the proof of Theorem 6. We have proved that Algorithm A creates a feasible schedule S_A (Proposition 4) in

polynomial time (Proposition 7) such that $c \cdot p_{sum} + Y(S_A) \leq (1 + 4\varepsilon/c)(c \cdot p_{sum} + Y(S^*))$ (Lemmas 3, 4, and 5), thus the theorem is proved. □

7.2. The combined method

In this section we show how to combine the methods of Sections 6.2 and 7.1 to get a PTAS for $P|d_j = d, n_i \leq N|c \cdot p_{sum} + Y$.

Theorem 7. *There is a PTAS for* $P|d_j = d, n_i \leq N|c \cdot p_{sum} + Y$.

Proof. By Theorems 6 and 4, we propose the following algorithm for $P|d_j = d, n_i \leq N|c \cdot p_{sum} + Y$.

Algorithm PTAS

Input: problem instance and parameter $\varepsilon \leq 1/3$.

1. Run Algorithm A and let S_A be the best schedule found.
2. Run Algorithm LPT, and let S_{LPT} be the schedule obtained.
3. If $Y(S_A) \leq Y(S_{LPT})$, then output S_A, else output S_{LPT}.

Since the conditions of Theorems 6 and 4 are complementary, it follows that Algorithm *PTAS* always outputs a solution of value at most $(1 + 4\varepsilon/c)$ times the optimum. The time complexity in either case is polynomial in the size of the input, hence, the algorithm is indeed a PTAS for our scheduling problem. □

Since our result is valid even if $N \geq n$, we have the following corollary:

Corollary 4. *There is a PTAS for* $P|d_j = d|c \cdot p_{sum} + Y$.

Notice that Theorem 1 remains valid if we replace Y by $c \cdot p_{sum} + Y$ in the late work minimization problem and $\tilde{Y}$ by $c \cdot a_{sum} + \tilde{Y}$ in the minimization variant of the resource leveling problem, thus we get the following by combining Theorems 1 and 7:

Corollary 5. *There is a PTAS for the resource leveling problem* $P|p_j = 1|c \cdot a_{sum} + \tilde{Y}$.

8. Final remarks

In this paper we have described a common approximation framework for 4 problems which have common roots. On the one hand, we have proposed the first polynomial time approximation scheme for the early work maximization problem on identical parallel machines with a common job due date when the number of the machines is part of the input, which generalizes the PTAS of Sterna and Czerniachowska (2017). Further on, we extended this result to the late work minimization problem, and to the maximization as well as the minimization variant of the resource leveling with unit time jobs problems. No approximation schemes were known for these problems before.

In the design of the PTAS for the early work maximization problem, we had some difficulties in showing the approximation guarantee. The technique we found may be used for designing (fully) polynomial time approximation schemes for completely different combinatorial optimization problems as well. We illustrate the main ideas for a maximization problem Π. Suppose we have devised a family of algorithms $\{A_\varepsilon\}_{\varepsilon > 0}$ for Π, but we are able to prove that it is a factor $(1 - \varepsilon)$ approximation algorithm only under the hypothesis that $OPT(I) \geq f(I, \varepsilon)$ for a problem instance I, where f is a function assigning some rational number to I and ε. Then we have to devise another algorithm, which is also a factor $(1 - \varepsilon)$ approximation algorithm on those instances such that $OPT(I) < f(I, \varepsilon)$. Now, if we run both methods on an arbitrary instance I, then at least one of them will return a solution of value at least $(1 - \varepsilon)$ times the optimum. Clearly, the combined method is an (F)PTAS for the problem Π.

There remained a number of open questions. For instance, is there a simple constant factor approximation algorithm for maximizing the early work on identical parallel machines with a common job due date, and has a running time suitable for practical applications? The same question can be asked for the late work minimization problem with the objective $c + Y$ for some positive c.

Acknowledgments

The authors are grateful to the anonymous referees for constructive comments that helped to improve the presentation.

References

Błażewicz, J. (1984). Scheduling preemptible tasks on parallel processors with information loss. *Technique et Science Informatiques, 3*(6), 415–420.
Błażewicz, J., Pesch, E., Sterna, M., & Werner, F. (2004). Open shop scheduling problems with late work criteria. *Discrete Applied Mathematics, 134*(1–3), 1–24.
Błażewicz, J., Pesch, E., Sterna, M., & Werner, F. (2005). The two-machine flow-shop problem with weighted late work criterion and common due date. *European Journal of Operational Research, 165*(2), 408–415.
Błażewicz, J., Pesch, E., Sterna, M., & Werner, F. (2007). A note on the two machine job shop with the weighted late work criterion. *Journal of Scheduling, 10*(2), 87–95.
Chen, X., Kovalev, S., Sterna, M., & Błażewicz, J. (2020a). Mirror scheduling problems with early work and late work criteria. *Journal of Scheduling, in press*. doi:10.1007/s10951-020-00636-9.
Chen, X., Liang, Y., Sterna, M., Wang, W., & Błażewicz, J. (2020b). Fully polynomial time approximation scheme to maximize early work on parallel machines with common due date. *European Journal of Operational Research, 284*(1), 67–74.
Chen, X., Sterna, M., Han, X., & Błażewicz, J. (2016). Scheduling on parallel identical machines with late work criterion: Offline and online cases. *Journal of Scheduling, 19*(6), 729–736. doi:10.1007/s10951-015-0464-7.
Drótos, M., & Kis, T. (2011). Resource leveling in a machine environment. *European Journal of Operational Research, 212*(1), 12–21. doi:10.1016/j.ejor.2011.01.043.
Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. New York, NY: Freeman.
Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics, 17*(2), 416–429.
Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326. doi:10.1016/S0167-5060(08)70356-X.
Kis, T. (2005). A branch-and-cut algorithm for scheduling of projects with variable-intensity activities. *Mathematical Programming, 103*(3), 515–539.
Kovalyov, M. Y., Potts, C. N., & Van Wassenhove, L. N. (1994). A fully polynomial approximation scheme for scheduling a single machine to minimize total weighted late work. *Mathematics of Operations Research, 19*(1), 86–93.
Kovalyov, M. Y., & Werner, F. (2002). Approximation schemes for scheduling jobs with common due date on parallel machines to minimize total tardiness. *Journal of Heuristics, 8*(4), 415–428.
Neumann, K., & Zimmermann, J. (2000). Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal of Operational Research, 127*(2), 425–443. doi:10.1016/S0377-2217(99)00498-1.
Potts, C. N., & Van Wassenhove, L. N. (1992a). Approximation algorithms for scheduling a single machine to minimize total late work. *Operations Research Letters, 11*(5), 261–266.
Potts, C. N., & Van Wassenhove, L. N. (1992b). Single machine scheduling to minimize total late work. *Operations Research, 40*(3), 586–595.
Rieck, J., Zimmermann, J., & Gather, T. (2012). Mixed-integer linear programming for resource leveling problems. *European Journal of Operational Research, 221*(1), 27–37.
Sterna, M. (2000). *Problems and algorithms in non-classical shop scheduling*. Scientific Publishers, Polish Academy of Sciences.
Sterna, M. (2006). *Late work scheduling in shop systems*. Dissertation 405. Poznań: Publishing House of Poznań University of Technology.
Sterna, M. (2007). Late work minimization in a small flexible manufacturing system. *Computers & Industrial Engineering, 52*(2), 210–228.
Sterna, M. (2011). A survey of scheduling problems with late work criteria. *Omega, 39*(2), 120–129. doi:10.1016/j.omega.2010.06.006.
Sterna, M., & Czerniachowska, K. (2017). Polynomial time approximation scheme for two parallel machines scheduling with a common due date to maximize early work. *Journal of Optimization Theory and Applications, 174*(3), 927–944. doi:10.1007/s10957-017-1147-7.
Verbeeck, C., Van Peteghem, V., Vanhoucke, M., Vansteenwegen, P., & Aghezzaf, E.-H. (2017). A metaheuristic solution approach for the time-constrained project scheduling problem. *OR Spectrum, 39*(2), 353–371.
Woeginger, G. J. (2005). A comment on scheduling two parallel machines with capacity constraints. *Discrete Optimization, 2*(3), 269–272.
Wright, S. J. (1997). Primal-dual interior-point methods. *Other Titles in Applied Mathematics*: 54. SIAM, Philadelphia, PA.