Egerváry Research Group
on Combinatorial Optimization

Technical reportS

TR-2015-16. Published by the Egerváry Research Group, Pázmány P. sétány 1/C, H–1117, Budapest, Hungary. Web site: www.cs.elte.hu/egres. ISSN 1587–4451.

Approximation schemes for parallel machine scheduling with non-renewable resources

Péter Györgyi and Tamás Kis

August 2015

Approximation schemes for parallel machine scheduling with non-renewable resources

Péter Györgyi and Tamás Kis*

Abstract

In this paper the approximability of parallel machine scheduling problems with resource consuming jobs is studied. In these problems, in addition to a parallel machine environment, there are non-renewable resources, like raw materials, energy, or money, consumed by the jobs. Each resource has an initial stock, and some additional supplies at a-priori known moments of times and in known quantities. The schedules must respect the resource constraints as well. The objective is the minimum schedule length or makespan. Polynomial time approximation schemes are provided under various assumptions, and it is shown that the problem becomes APX-hard if the number of machines is part of the input even if there are only two resources.

Keywords: Scheduling, parallel machines, non-renewable resources, approximation schemes

1 Introduction

In this paper we study various parallel machine scheduling problems with resource consuming jobs. In these problems, in addition to machines, there are non-renewable resources (like raw materials, energy, or money) consumed by the jobs. Each non-renewable resource has an initial stock, which is replenished at a-priori known moments of time and in known quantities. We will consider two types of parallel machine environments: (i) the standard parallel machine environment, where each job can be scheduled on any machine, and the job processing times do not depend on the machines assigned, and (ii) the one with dedicated machines, where each job can be executed only on one of the machines. In all problems considered the machines can perform only one job at a time, and preemption of jobs is not allowed. The objective is

*Department of Operations Research, Loránd Eötvös University, H1117 Budapest, Pázmány Péter sétány 1/C, Hungary and Institute for Computer Science and Control, H1111 Budapest, Kende str. 13–17, Hungary. E-mails: gyorgyip@cs.elte.hu (Péter Györgyi), kis.tamas@sztaki.mta.hu (Tamás Kis)

to minimize the maximal completion time of the jobs, or in other words the *makespan* of the schedule.

More formally, there are m parallel machines, $\mathcal{M} = \{M_1, \ldots, M_m\}$, a finite set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, and a finite set of non-renewable resources $\mathcal{R}$ consumed by the jobs. Each job J_j has a processing time $p_j \in \mathbb{Z}_+$, a release date r_j, and resource requirements $a_{ij} \in \mathbb{Z}_+$ from the resources $i \in \mathcal{R}$. The resources are supplied in q different time moments, $0 = u_1 < u_2 < \ldots < u_q$; the vector $\tilde{b}_\ell \in \mathbb{Z}_+^{|\mathcal{R}|}$ represents the quantities supplied at u_ℓ. A *schedule* σ specifies a machine and the starting time S_j of each job and it is *feasible* if (i) in every machine the jobs do not overlap in time, (ii) $S_j \geq r_j$ for all $j \in \mathcal{J}$, and if (iii) at any time point t the total material supply from every resource is at least the total request of those jobs starting not later than t, i.e., $\sum_{(\ell \,:\, u_\ell \leq t)} \tilde{b}_{\ell i} \geq \sum_{(j \,:\, S_j \leq t)} a_{ij},\ \forall i \in \mathcal{R}$. The objective is to minimize the makespan, i.e., the completion time of the job finished last.

Assumption 1. $\sum_{\ell=1}^{q} \tilde{b}_{\ell i} = \sum_{j \in \mathcal{J}} a_{ij},\ \forall i \in \mathcal{R}$, *holds without loss of generality.*

Since the makespan minimization problem with resource consuming jobs on a single machine is NP-hard even if there are only two supply dates [3], all problems studied in this paper are NP-hard.

The combination of scheduling and logistic, that is, considering e.g., raw material supplies in the course of scheduling, has a great practical potential, as this problem frequently occurs in practice according to the experience of the authors.

1.1 Main results

If the number of the machines is part of the input, then we have the following non-approximability result:

Theorem 1.1. *Deciding whether there is a schedule of makespan 2 with two non-renewable resources, two supply dates and unit-time jobs on arbitrary number of machines ($P|rm = 2, q = 2, p_j = 1|C_{\max} \leq 2$) is NP-hard. Consequently, it is NP-hard to approximate this problem better than $3/2 - \varepsilon$ for any $\varepsilon > 0$.*

By assumption 1, the optimum makespan is at least u_q, therefore, a straightforward two-approximation algorithm would schedule all the jobs after u_q. Therefore, we have the following

Corollary 1.2. $P|rm = 2, q = 2, p_j = 1|C_{\max}$ *is APX-hard.*

In case of constant number of machines, we provide polynomial time approximation schemes (PTAS) for various special cases of $Pm|rm|C_{\max}$. In all cases, we allow that jobs have distinct release dates. Note that the problem with arbitrary number of resources is APX-hard even in case of a single machine ([10]), thus we need the constraint $rm = const.$

If the number of resources is a constant, then there is a PTAS under the condition that the number of machines is fixed:

Theorem 1.3. $Pm|rm = const., q = const., r_j|C_{\max}$ *admits a PTAS.*

If the jobs are dedicated to machines we have an analogous result:

Theorem 1.4. *$Pm|rm = const., q = const., r_j, ddc|C_{\max}$ admits a PTAS.*

Since $P|ddc|C_{\max}$ and thus $Pm|ddc|C_{\max}$ are trivially solvable in polynomial time, the complexity of the this problem stems from the addition of non-renewable resource constraints.

In case of a single machine, if there is a single resource and a constant $\lambda > 0$ such that $a_j = \lambda p_j$, then we have a PTAS for the makespan minimization problem even if the number of supply dates is arbitrary [10]. Luckily, this property carries over to the more general parallel machine case. The constant λ of course depends on the problem instance. This assumption may be quite reasonable in some practical applications. Since we can get an equivalent problem by dividing all the supplies, and all the resource requirements of a problem instance by the (instance specific) constant λ, from now on we consider the case $a_j = p_j$ only. Notice that in the above transformation, the $\tilde{b}_\ell$ may become fractional after dividing by λ. However, this does not create any difficulty for the approximation algorithm proposed below.

Theorem 1.5. *$Pm|rm = 1, a_j = p_j, r_j|C_{\max}$ admits a PTAS.*

We can deal with dedicated jobs as before:

Theorem 1.6. *$Pm|rm = 1, a_j = p_j, r_j, ddc|C_{\max}$ admits a PTAS.*

1.2 Structure of the paper

In Section 2 we summarize previous work on machine scheduling with non-renewable resources. In Section 3 we provide a problem formulation in terms of mathematical program which will be used throughout the paper. In Sections 4, 5, 6, 7 and 8 we prove Theorems 1.1, 1.3, 1.4, 1.5 and 1.6, respectively.

1.3 Terminology

An *optimization problem* Π consists of a set of instances, where each instance has a set of *feasible solutions*, and each solution has a cost. In a *minimization problem* a feasible solution of minimum cost is sought, while in a maximization problem one of maximum cost. An *ε-approximation algorithm* for an optimization problem Π delivers in polynomial time for each instance of Π a solution whose objective function value is at most $(1+\varepsilon)$ times the optimum value in case of minimization problems, and at least $(1-\varepsilon)$ times the optimum in case of maximization problems. For an optimization problem Π, a family of approximation algorithms $\{A_\varepsilon\}_{\varepsilon>0}$, where each A_ε is an ε-approximation algorithm for Π is called a *Polynomial Time Approximation Scheme (PTAS) for* Π.

2 Previous work

Scheduling problems with resource consuming jobs were introduced by [3], [4], and [16]. In [3], the computational complexity of several variants with a single machine was established, while in [4] activity networks requiring only non-renewable resources were considered. In [16] a parallel machine problem with preemptive jobs was studied, and the single non-renewable resource had an initial stock and some additional supplies, like in the model presented above, and it was assumed that the rate of consuming the non-renewable resource was constant during the execution of the jobs. These assumptions led to a polynomial time algorithm for minimizing the makespan, which is in strong contrast to the NP-hardness of all the scheduling problems analyzed in this paper. Further results can be found in e.g., [17], [15], [13], [8], [1], [2], [6], [9], [11], [10], [14]. In particular, [17] proved that scheduling jobs requiring one non-renewable resource on a single machine with the objective of minimizing the makespan reduces to the 2-machine flow shop problem provided that the single non-renewable resource has a unit supply in every time period. [15] study general project scheduling problems with inventory constraints, and propose a branch-and-bound algorithm for minimizing the project length. In a more general setting, jobs may consume as well as produce non-renewable resources. In [8] and [6] the complexity of several variants was studied and some constant ratio approximation algorithms were developed in [8]. [1], [2] and [14] examined scheduling problems where there is an initial inventory, and no more supplies, but some of the jobs produce resources, while other jobs consume the resources. In [1] and [2] scheduling problems with the objective of minimizing the inventory levels were studied. [14] designed approximation algorithms to minimize the total weighted completion time. In [9] a PTAS for scheduling resource consuming jobs with a single non-renewable resource and a constant number of supply dates was developed, and also an FPTAS was devised for the special case with $q = 2$ supply dates and one non-renewable resource only. In [11] it was shown, among other results, that there is no FPTAS for the problem of scheduling jobs on a single machine with two non-renewable resources and $q = 2$ supply dates, unless $P = NP$, which is in strong contrast with the existence of an FPTAS for the special case with one non-renewable resource only [9]. These results have been extended in [10]: it contains a PTAS under various assumptions: (1) both the number of resources and the number of supplies dates are constants, (2) there is only one resource, arbitrary number of supply dates, but the resource requirements are proportional to job processing times. It also proves the APX-hardness of the problem when the number of resources is part of the input. In Table 1 we summarize the known and new results of scheduling resource consuming jobs in single machine as well as in parallel machine environments, when preemption of processing is not allowed, and the resources are consumed right at starting the jobs.

3 A mathematical program

We can model $P|rm|C_{\max}$ with a mathematical program with integer variables. Let $\mathcal{M}$ denote the set of the machines and let $\mathcal{T}$ be the union of the set of supply dates and job

#Machines m	#Supplies q	#Resources rm	Release dates r_j	PTAS	FPTAS
1	2	1	no	yes ([9])	yes ([9], [11])
1	2	1	yes	yes ([10])	?
1	2	*const.* ≥ 2	no	yes ([9])	no ([11])
1	2	*const.* ≥ 2	yes	yes ([10])	no ([11])
1	2	arbitrary	yes/no	no ([10])	no ([10])
1	*const.* ≥ 3	1	yes/no	yes ([10])	?
1	*const.* ≥ 3	*const.* ≥ 2	yes/no	yes ([10])	no ([11])
1	arbitrary	1*	yes/no	yes ([10])	no ([8])
const ≥ 2	*const.* ≥ 3	*const.* ≥ 2	yes/no	yes (Sect. 5)	no ([8])
const ≥ 2	arbitrary	1*	yes/no	yes (Sect. 7)	no ([8])
arbitrary	2	2	yes/no	no (Sect. 4)	no ([8])

* under the condition $a_j = \lambda p_j$

Table 1: Known approximability results for scheduling problems with resource consuming jobs if $\mathcal{P} \neq \mathcal{NP}$. In the column of Release dates "yes / no" means that the result is valid in both cases. The question mark "?" indicates that we are not aware of any definitive answer.

release dates, i.e., $\mathcal{T} := \{u_\ell \mid \ell = 1, \ldots, q\} \cup \{r_j \mid j \in \mathcal{J}\}$. Suppose $\mathcal{T}$ has τ elements, denoted by v_1 through v_τ, with $v_1 = 0$. We define the values $b_{\ell i} := \sum_{\nu \,:\, u_\nu \leq v_\ell} \tilde{b}_{\nu i}$ for $i \in \mathcal{R}$, that is, $b_{\ell i}$ equals the total amount supplied from resource i up to time point v_ℓ.

We introduce $\tau \cdot |\mathcal{J}||\mathcal{M}|$ binary decision variables $x_{j\ell k}$, $(j \in \mathcal{J}, \ell = 1, \ldots, \tau, k \in \mathcal{M})$ such that $x_{j\ell k} = 1$ if and only if job j is assigned to machine k and to the time point v_ℓ, which means that the requirements of job j must be satisfied by the resource supplies up to time point v_ℓ. The mathematical program is

$$C^*_{\max} = \min \max_{k \in \mathcal{M}} \max_{v_\ell \in \mathcal{T}} \left(v_\ell + \sum_{j \in \mathcal{J}} \sum_{\nu=\ell}^{\tau} p_j x_{j\nu k} \right) \tag{1}$$

s.t.

$$\sum_{k \in \mathcal{M}} \sum_{j \in \mathcal{J}} \sum_{\nu=1}^{\ell} a_{ij} x_{j\nu k} \leq b_{\ell i}, \quad v_\ell \in \mathcal{T},\ i \in \mathcal{R} \tag{2}$$

$$\sum_{k \in \mathcal{M}} \sum_{\ell=1}^{\tau} x_{j\ell k} = 1, \quad j \in \mathcal{J} \tag{3}$$

$$x_{j\ell k} = 0, \quad j \in \mathcal{J},\ v_\ell \in \mathcal{T} \text{ such that } r_j > v_\ell,\ k \in \mathcal{M} \tag{4}$$

$$x_{j\ell k} \in \{0, 1\}, \quad j \in \mathcal{J},\ v_\ell \in \mathcal{T},\ k \in \mathcal{M}. \tag{5}$$

The objective function expresses the completion time of the job finished last using the observation that for every machine there is a time point, either a release date of

some job, or when some resource is supplied from which the machine processes the jobs without idle times. Constraints (2) ensure that the jobs assigned to time points v_1 through v_ℓ use only the resources supplied up to time v_ℓ. Equations (3) ensure that all jobs are assigned to some machine and time point. Finally, no job may be assigned to a time point before its release date by (4). Any feasible job assignment $\bar{x}$ gives rise to a set of schedules which differ only in the ordering of jobs assigned to the same machine k, and time point v_ℓ.

Notice that in a feasible solution $\hat{x}$ of (1)-(5) there can be more than one jobs assigned to the same machine k and time point v_ℓ. We obtain a schedule of the jobs by putting them on machine k in the order of their assignment to the time points in $\mathcal{T}$ (we do the same for every $k \in \mathcal{M}$). That is, first we schedule in non-increasing processing time order without idle times the jobs with $\hat{x}_{j1k} = 1$ from time v_1 on. Let $C_1(k)$ be the completion time of these jobs. In a general step $\ell \geq 2$, we schedule the jobs with $\hat{x}_{j\ell k} = 1$ in non-increasing processing time order after $\max\{C_{\ell-1}(k), v_\ell\}$, and we denote by $C_\ell(k)$ the completion time of the job finished last in this group on machine k. The non-increasing processing time order is due to technical reasons, the makespan of a schedule is the same in case of any order. The schedule obtained in this way is feasible, and its makespan is the completion time of the job finished last, which is necessarily equal to the objective function value of solution $\hat{x}$. Let $C_{\max}(\hat{x})$ denote this value.

Sometimes we need to create a (partial) assignment from a (partial) schedule. That is the following: if J_j is scheduled on M_k at S_j and $v_\ell \leq S_j < v_{\ell+1}$, then let $x_{j\ell k} = 1$ and $x_{j\ell' k'} = 0$ if $(\ell', k') \neq (\ell, k)$. Notice that, the value of this (partial) assignment is at most the makespan of the (partial) schedule.

Note that the number of the different assignments is at most $O((m\tau)^n)$. If $\varepsilon \leq 1/n$ in a PTAS, then we can check every assignment in $O((m\tau)^n) \leq O((m\tau)^{1/\varepsilon})$ time, which is polynomial in the size of the input, thus from now we assume

Assumption 2. $\varepsilon > 1/n$.

4 APX-hardness of $P|rm = 2, q = 2, p_j = 1|C_{\max}$

In this section we prove Theorem 1.1. We reduce the EVEN-PARTITION problem to the problem $P|rm = 2, q = 2, p_j = 1|C_{\max}$, and argue that deciding whether a schedule of makespan two exists is as hard as finding a solution for EVEN-PARTITION. Recall that an instance of the EVEN-PARTITION problem consists of $2t$ items, for some integer t, of sizes $a_1, \ldots, a_{2t} \in \mathbb{Z}_+$. The decision problem asks whether there is a subset of items S of cardinality t such that $\sum_{i \in S} a_i = \sum_{\bar{S}} a_i$? This problem is NP-hard in the ordinary sense, see [7]. Clearly, a necessary condition for the existence of set S is that the total size of all items is an even integer, i.e., $\sum_{i=1}^{2t} a_i = 2A$, for some $A \in \mathbb{Z}_+$.

Proof of Theorem 1.1 We map an instance I of EVEN-PARTITION to the following instance of $P|rm = 2, q = 2, p_j = 1|C_{\max}$. There are $n := 2t$ jobs, and $m := t$ machines. All the jobs have unit processing time, i.e., $p_j = 1$ for all j. The

job corresponding to the jth item in I has resource requirements $a_{1,j} := a_j$ and $a_{2,j} := A - a_j$. The initial supply at $u_1 = 0$ from the two resources is $\tilde{b}_{1,1} := A$ and $\tilde{b}_{1,2} := (t-1)A$, and the second supply at time $u_2 = 1$ has $\tilde{b}_{2,1} := A$, and $\tilde{b}_{2,2} := (t-1)A$. We have to decide whether a feasible schedule of makespan two exists.

First, suppose that I has a solution S. Then we schedule all the jobs corresponding to the items in S at time 0, each on a separate machine. Since S contains t items, and the number of machines is t as well, this is feasible. Moreover, the total resource requirement from the first resource is precisely A, whereas that from the second one is $\sum_{j\in S} a_{2,j} = \sum_{j\in S}(A - a_j) = (t-1)A$. The rest of the jobs are scheduled at time 1. Since their number is t, and since $u_2 = 1$ is the second and last supply date, all the resources are supplied and the jobs can start promptly at time 1.

Conversely, suppose there is a feasible schedule of makespan two. Then, there are t jobs scheduled at time 0, and the remaining t jobs at time 1. The resource requirements of those jobs scheduled at time 0 equal the supply at time $u_1 = 0$, hence these jobs define set S, a feasible solution of the EVEN-PARTITION problem instance. □

5 PTAS for $Pm|rm = const, q = const, r_j|C_{\max}$

In this section we prove Theorem 1.3. First, note that we can simplify the problem: it is enough to deal with the case where the number of distinct release dates is a constant. There are several techniques that we can use for this simplification, see e.g. section 5 in [10] or section 2 in [12].

Let $p_{\text{sum}} := \sum_{j\in\mathcal{J}} p_j$ and note that $p_{\text{sum}} \le mC^*_{\max}$. For a fixed $\varepsilon > 0$, let $\mathcal{B} := \{j \in \mathcal{J} | p_j \ge \varepsilon p_{\text{sum}}\}$ be the set of big jobs, and $\mathcal{S} := \mathcal{J} \setminus \mathcal{B}$ be the set of small jobs. We divide further the set of small jobs according to their release dates, that is, we define the sets $\mathcal{S}^b := \{j \in \mathcal{S} \mid r_j < u_q\}$, and $\mathcal{S}^a := \mathcal{S} \setminus \mathcal{S}^b$. Let $\mathcal{T}^b := \{v_\ell \in \mathcal{T} \mid v_\ell < u_q\}$ be the set of time points v_ℓ before u_q. The following observation reduces the number of solutions of (1)-(5) to be examined.

Proposition 5.1. *From any feasible solution $\hat{x}$ of (1)-(5), we can obtain a solution $\tilde{x}$ with $C_{\max}(\tilde{x}) \le C_{\max}(\hat{x})$ such that each job J_j is assigned to some time point v_ℓ ($\sum_{k\in\mathcal{M}} \tilde{x}_{j\ell k} = 1$), satisfying either $v_\ell < u_q$, or $v_\ell = \max\{u_q, r_j\}$.*

The above statement is a generalization of the single machine case treated in [10], and its proof can be found in the Appendix.

An *assignment of big jobs* is given by a partial solution $x^{big} \in \{0,1\}^{\mathcal{B}\times\mathcal{T}\times\mathcal{M}}$ which assigns each big job to some machine k and time point v_ℓ. An assignment x^{big} of big jobs is *feasible* if the vector $x = (x^{big}, 0) \in \{0,1\}^{\mathcal{J}\times\mathcal{T}\times\mathcal{M}}$ satisfies (2), (4) and also (3) for the big jobs. Consider any feasible assignment x^{big} of big jobs. If we fix the assignment of the big jobs in (2)-(4) to x^{big}, then the supply from any resource i up to time point v_ℓ is decreased by the requirements of those big jobs assigned to time points v_1 through v_ℓ. Hence, we define the *residual resource supply* up to time point v_ℓ as $\bar{b}_{\ell i} := b_{\ell i} - \sum_{k\in\mathcal{M}} \sum_{j\in\mathcal{B}} a_{ij} \left(\sum_{\nu=1}^{\ell} x^{big}_{j\nu k}\right)$. Further on, let $\bar{C}^{\mathcal{B}}_\ell(k) :=$

$\max_{\nu=1,\dots,\ell}(v_\nu + \sum_{\kappa=\nu}^{\ell}\sum_{j\in\mathcal{B}} p_j x^{big}_{j\kappa k})$ denote the earliest time point when the big jobs assigned to v_1 through v_ℓ may finish on machine k. Notice that $\bar{C}^{\mathcal{B}}_\ell(k) \geq v_\ell$ even if no big job is assigned to v_ℓ, or to any time period before v_ℓ.

In order to assign approximately the small jobs, we will solve a linear program and round its solution. Our linear programming formulation relies on the following result.

Proposition 5.2. *There exists an optimal solution $(\hat{x}^{big}, \hat{x}^{small})$ of (1)-(5) such that for each $v_\ell \in \mathcal{T}^b,\ k \in \mathcal{M}$:*

$$\sum_{j\in\mathcal{S}^b} p_j \hat{x}^{small}_{j\ell k} \leq \max\{0, v_{\ell+1} - \bar{C}^{\mathcal{B}}_\ell(k)\} + \varepsilon p_{\text{sum}}. \tag{6}$$

The above statement is a generalization of the single machine case treated in [10], and its proof can be found in the Appendix.

For every feasible big job assignment we will determine a complete solution of (1)-(5). We search these solution in two steps: first we assign the small jobs to time moments and then to machines. Let $x_{j\ell} := \sum_{k\in\mathcal{M}} x_{j\ell k}$. Now, the linear program is defined with respect to any feasible assignment x^{big} of the big jobs:

$$\max \sum_{v_\ell\in\mathcal{T}^b}\sum_{j\in\mathcal{S}^b} p_j x^{small}_{j\ell} \tag{7}$$

s.t.

$$\sum_{j\in\mathcal{S}^b}\sum_{\nu=1}^{\ell} a_{ij} x^{small}_{j\nu} \leq \bar{b}_{\ell i}, \quad v_\ell \in \mathcal{T}^b,\ i \in \mathcal{R} \tag{8}$$

$$\sum_{j\in\mathcal{S}^b} p_j x^{small}_{j\ell} \leq \sum_k \max\{0, v_{\ell+1} - \bar{C}^{\mathcal{B}}_\ell(k)\} + m\varepsilon p_{\text{sum}}, \quad v_\ell \in \mathcal{T}^b \tag{9}$$

$$\sum_{v_\ell\in\mathcal{T}^b\cup\{u_q\}} x^{small}_{j\ell} = 1, \quad j \in \mathcal{S}^b \tag{10}$$

$$x^{small}_{j\ell} = 0, \quad j \in \mathcal{S}^b,\ v_\ell \in \mathcal{T} \text{ such that } v_\ell < r_j, \text{ or } v_\ell > u_q \tag{11}$$

$$x^{small}_{j\ell} \geq 0, \quad j \in \mathcal{S}^b,\ v_\ell \in \mathcal{T}. \tag{12}$$

The objective function (7) maximizes the total processing time of those small jobs assigned to some time point v_ℓ *before* u_q. Constraints (8) make sure that no resource is overused taking into account the fixed assignment of big jobs as well. Inequalities (9) ensure that the total processing time of those small jobs assigned to $v_\ell \in \mathcal{T}^b$ does not exceed the total size of all the gaps on the m machines between v_ℓ and $v_{\ell+1}$ by more than $m\varepsilon p_{\text{sum}}$. Due to (10), small jobs are assigned to some time point in $\mathcal{T}^b \cup \{u_q\}$. The release dates of those jobs in $\mathcal{S}^b$, and Proposition 5.1 are taken care of by (11). Finally, we require that the values $x^{small}_{j\ell}$ be non-negative.

Notice that this linear program always has a finite optimum provided that x^{big} is a feasible assignment of the big jobs. Let $\bar{x}^{small}$ be any feasible solution of the linear program. Job $j \in \mathcal{S}^b$ is *integral* in $\bar{x}^{small}$ if there exists $v_\ell \in \mathcal{T}$ with $\bar{x}^{small}_{j\ell} = 1$, otherwise it is *fractional*. After all these preliminaries, the PTAS is as follows.

Algorithm A

1. Assign the big jobs to time points v_1 through v_τ and to machines 1 through $|\mathcal{M}|$ in all possible ways which satisfies Proposition 5.1, and for each feasible assignment x^{big} do steps 2 - 7 :

2. Define and solve linear program (7)-(12), and let $\bar{x}^{small}$ be an optimal basic solution.

3. Round each fractional value in $\bar{x}^{small}$ down to 0, and let $x^{small} := \lfloor \bar{x}^{small} \rfloor$ be the resulting partial assignment of small jobs, and $U \subset \mathcal{S}^b$ the set of fractional jobs in $\bar{x}^{small}$.

4. The next algorithm assigns a machine for every small job that is assigned to a time point before u_q. For each $\ell \in \mathcal{T}^b$ do:

   i) Put the small jobs with $\bar{x}^{small}_{j\ell} = 1$ into a list in an arbitrary order.

   ii) For $k = 1, \ldots, m$ do the following steps:

      a) Let t be such that the total processing time of the first t jobs from the ordered list is in $[\max\{0, v_{\ell+1} - \bar{C}^{\mathcal{B}}_\ell(k)\} + \varepsilon p_{\text{sum}}, \max\{0, v_{\ell+1} - \bar{C}^{\mathcal{B}}_\ell(k)\} + 2\varepsilon p_{\text{sum}}]$. If there is no such t, then let t be the number of the small jobs in the ordered list.

      b) Assign the first t jobs to machine k and delete them from the ordered list.

5. Create a (partial) schedule S^{part} of the jobs that have already been assigned to a time period and a machine. Let $C^{part}_{\max}$ denote the makespan of this schedule.

6. Order the remaining jobs into a non-decreasing order of their release dates $(J_1, J_2, \ldots)$. We will schedule these jobs one by one. Let J_j be the next job to be scheduled. Let M_k be a machine with the earliest idle time after $\max\{u_q, r_j\}$ in the current schedule. Schedule J_j on this machine at that time. If necessary, push to the right the later jobs and proceed with the next unscheduled job.

7. If the makespan of the complete schedule of all the jobs is better than the best solution found so far, then update the best solution.

8. After examining each feasible assignment of the big jobs, output the best complete solution found.

See Figure 1 for illustration. We will prove that the solution found by the previous algorithm is feasible for (1)-(5), its value is not far from the optimum, and the algorithm runs in polynomial time.

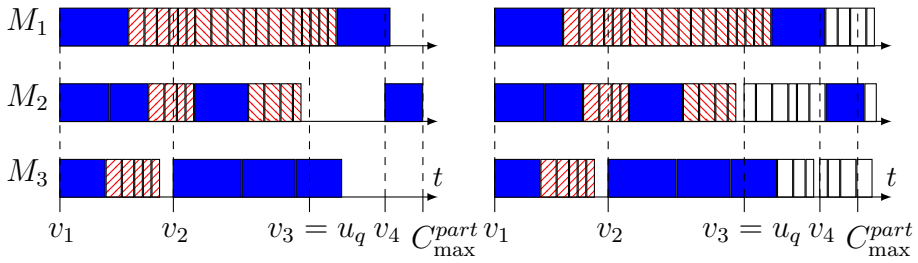

Figure 1: A partial schedule on the left (big jobs are blue, small jobs are hatched) and a complete schedule on the right. The jobs scheduled at Step 6 are white. Each job scheduled after v_4 has a release date v_4, since M_3 is idle before v_4.

Lemma 5.3. *Every complete solution (x^{big}, x^{small}) constructed by the algorithm is feasible for (1)-(5).*

Proof. At the end of the algorithm each job is scheduled exactly once sometime after its release date, thus the solution satisfies 3, 4 and 5. The algorithm examines only feasible assignments of the big jobs, hence these jobs cannot hurt the resource constraints. Since $\bar{x}^{small}$ is a feasible solution of 7 - 12 and $\sum_{k\in\mathcal{M}} x_{j\ell k} = x_{j\ell}$, $(\forall j \in \mathcal{J})$, thus the assignment corresponds to S^{part} satisfies 2. Finally, since u_q is the last time point when some resource is supplied, thus when the algorithm schedules the remaining jobs at Step 6, the constraints 2 remain feasible. □

To prove that the makespan of the schedule found by the algorithm is near to the optimum, we need Propositions 5.4 and 5.5. From these we conclude that the fractionally assigned jobs and the 'errors' in (9) do not cause big delays. We utilize that the number of the release dates before u_q is a constant. From the second proposition, we can deduce that, in case of appropriate big job assignment, $C^{part}_{\max}$ is not much bigger than $C^*_{\max}$. If the found makespan is larger than $C^{part}_{\max}$, then the machines finish the assigned jobs nearly at the same time, thus we can prove that there are no big delays relative to an optimal solution in the solution found by the algorithm.

Proposition 5.4. *In any basic solution of the linear program (7)-(12), there are at most $(|\mathcal{R}| + 1) \cdot |\mathcal{T}^b|$ fractional jobs.*

Proof. Let $\bar{x}^{small}$ be a basic solution of the linear program in which f jobs of $\mathcal{S}^b$ are assign fractionally, and $e = |\mathcal{S}^b| - f$ jobs integrally. Clearly, each integral job gives rise to precisely one positive value, and each fractionally assigned job to at least two. This program has $|\mathcal{S}^b| \cdot |\mathcal{T}^b|$ decision variables, and $\gamma = |\mathcal{S}^b| + (|\mathcal{R}| + 1) \cdot |\mathcal{T}^b|$ constraints. Therefore, in $\bar{x}^{small}$ there are at most γ positive values, as no variable may be nonbasic with a positive value. Hence,

$$e + 2f \leq |\mathcal{S}^b| + (|\mathcal{R}| + 1) \cdot |\mathcal{T}^b| = e + f + (|\mathcal{R}| + 1) \cdot |\mathcal{T}^b|.$$

This implies

$$f \leq (|\mathcal{R}| + 1) \cdot |\mathcal{T}^b|$$

as claimed. □

Proposition 5.5. *Consider a big job assignment after Step 1, let S^{big} denote the partial schedule of this assignment and $C^{\mathcal{B}}_{\max}$ denote its makespan.*

1. *If a big job J_j is assigned to v_ℓ at Step 1, then $S_j^{part} \leq S_j^{big} + 2(\min\{\ell - 1, |\mathcal{T}^b|\})\varepsilon p_{\text{sum}}$.*
2. *$C_{\max}^{part} \leq \max\{u_q, C^{\mathcal{B}}_{\max}\} + 2|\mathcal{T}^b|\varepsilon p_{\text{sum}}$.*

Proof. Recall that the jobs assigned to the same time point and machine are in non-increasing processing time order.

1. The algorithm can push to the right a start time of big job at Step 4(ii)a, or in other words, when it assigns small jobs before v_ℓ. At every turn, this step pushes to the right J_j with at most $2\varepsilon p_{\text{sum}}$. This event happens $\min\{\ell - 1, |\mathcal{T}^b|\}$ times, thus the proposition follows.
2. Imagine a fictive big job starts at $\max\{u_q, C^{\mathcal{B}}_{\max}\}$, and apply the first part of the proposition.

□

Lemma 5.6. *The algorithm constructs at least one complete assignment (x^{big}, x^{small}) whose value is at most $(1 + O(\varepsilon))$ times the optimum makespan $C^*_{\max}$.*

Proof. Consider an optimal schedule $S^* = (\hat{x}^{big}, \hat{x}^{small})$ of (1)-(5) that satisfies Proposition 5.2. The algorithm will examine $\hat{x}^{big}$, since it is a feasible big job assignment. Let $C_{\max}$ denote the makespan of the schedule S found by the algorithm in this case. The observation below follows from Proposition 5.5:

Observation 5.7. $C_{\max}^{part} \leq C^*_{\max} + 2|\mathcal{T}^b|\varepsilon p_{\text{sum}}$.

If no small job scheduled at Step 6 starts after $C_{\max}^{part} - \varepsilon p_{\text{sum}}$, then the statement of the lemma follows from Observation 5.7 since $p_{\text{sum}} \leq mC^*_{\max}$ and $C_{\max} \leq C_{\max}^{part} + \varepsilon p_{\text{sum}}$, thus $C_{\max} \leq (1 + (2|\mathcal{T}^b| + 1)m\varepsilon)C^*_{\max}$.

From now on, suppose that at least one small job scheduled at Step 6 starts after $C_{\max}^{part} - \varepsilon p_{\text{sum}}$. For similar reasons, also suppose that $C_{\max} > \max\{C_{\max}^{part}, v_\tau\} + \varepsilon p_{\text{sum}}$ (this means that for every machine there is at least one small job that starts after $\max\{C_{\max}^{part}, v_\tau\}$ and scheduled at Step 6).

Observation 5.8. The difference between the finishing time of two arbitrary machines is at most $\varepsilon p_{\text{sum}}$.

We prove the statement of the lemma with Claims 1, 2 and 3.

Claim 1. *If there is no gap on any machine, then $C_{\max} \leq (1 + m\varepsilon)C^*_{\max}$.*

Proof. According to Observation 5.8 each machine is working between 0 and $(C_{\max} - \varepsilon p_{\text{sum}})$. Therefore $C^*_{\max} \geq C_{\max} - \varepsilon p_{\text{sum}}$ which implies $C_{\max} \leq (1 + m\varepsilon)C^*_{\max}$. □

Claim 2. *If the last gap finishes after u_q, then $C_{\max} \leq (1 + (2|\mathcal{T}^b| + 1)m\varepsilon)C^*_{\max}$.*

Proof. Note that this gap must finish at a release date r_{j_0}. Notice that each small job scheduled after r_{j_0} has a release date at least r_{j_0} or else we would have scheduled that job into the last gap, thus

Observation 5.9. The small jobs starting after r_{j_0} in S are scheduled after r_{j_0} in S^*.

Consider an arbitrary machine M_k and the last big job J_j that is starting before r_{j_0} on this machine in S^*. If $S_j^{part} < u_q$ or there is no gap between u_q and S_j^{part} in S^{part}, then we have not scheduled any job on M_k before J_j at Step 6, thus the starting (and the completion) time of J_j is at most $2|\mathcal{T}^b|\varepsilon p_{\text{sum}}$ later in S than in S^* (Proposition 5.5). Otherwise the starting time of J_j is the same in S^{part} and in S^* ($S_j^{part} = S_j^*$), since we can suppose that the jobs assigned to the same time point and machine are scheduled in the same non-increasing processing time order. If we push S_j at Step 6 once, then we cannot schedule any more jobs before S_j later, thus we can push S_j by at most $\varepsilon p_{\text{sum}}$ in total, thus

Observation 5.10. If $J_j \in \mathcal{B}$, then $S_j \leq S_j^* + 2|\mathcal{T}^b|\varepsilon p_{\text{sum}}$.

Suppose that a job J_j is scheduled from S'_j to $C'_j = S'_j + p_j$ in a schedule S' and $S'_j \leq t \leq C'_j$. In this case we can divide J_j into two parts: to the part of J_j that is scheduled before t (it has a processing time of $t - S'_j$) and to the part that is scheduled after t (it has a processing time of $C'_j - t$). Suppose that t is fixed and we divided all the jobs such that $S'_j \leq t \leq C'_j$ into two parts. Let $P_b^{(t)}(S')$ denote the total processing time of the jobs and job parts that are scheduled before t in S' and $P_a^{(t)}(S')$ denote the same after t ($P_b^{(t)}(S') + P_a^{(t)}(S') = p_{\text{sum}}$).

Observation 5.11. $P_a^{(r_{j_0}+2|\mathcal{T}^b|\varepsilon p_{\text{sum}})}(S) \leq P_a^{(r_{j_0})}(S^*)$ (follows from Observations 5.9 and 5.10).

Let $P := P_a^{(r_{j_0}+2|\mathcal{T}^b|\varepsilon p_{\text{sum}})}(S)$. Since there is no gap after r_{j_0} in S, $C_{\max} \leq r_{j_0} + 2|\mathcal{T}^b|\varepsilon p_{\text{sum}} + (P/m + \varepsilon p_{\text{sum}})$ follows from Observation 5.8. Since $C^*_{\max} \geq r_{j_0} + P/m$ (from Observation 5.11), thus $C_{\max} \leq C^*_{\max} + (2|\mathcal{T}^b| + 1)\varepsilon p_{\text{sum}} \leq (1 + (2|\mathcal{T}^b| + 1)m\varepsilon)C^*_{\max}$, therefore we have proved Claim 2. □

For a schedule S', let $S'_{\mathcal{B}}$ denote the schedule of the big jobs (where the big jobs have the same starting times as in S' and the small jobs are deleted from S') and $S'_{\mathcal{S}}$ denote the schedule of the small jobs (similarly).

Claim 3. *If each gap finishes before u_q, then $C_{\max} \leq (1 + ((2|\mathcal{T}^b| + 1)m + (|\mathcal{R}| + 1) \cdot |\mathcal{T}^b|)\varepsilon)C^*_{\max}$.*

Proof. Note that, each machine is working between u_q and $C_{\max} - \varepsilon p_{\text{sum}}$. Since $\bar{x}^{small}$ is an optimal solution of (7)-(12) and according to Proposition 5.2 $\hat{x}^{small}$ is a feasible solution, thus $\sum_{j \in \mathcal{S}: S_j^* \leq u_q} p_j \leq \sum_{j \in \mathcal{S}: \text{scheduled at Step 5}} p_j + \sum_{j \in U} p_j$, therefore $P_b^{(u_q)}(S^*_{\mathcal{S}}) \leq P_b^{(u_q+2|\mathcal{T}^b|\varepsilon p_{\text{sum}})}(S_{\mathcal{S}}) + \sum_{j \in U} p_j$ (Proposition 5.5). $P_b^{(u_q)}(S^*_{\mathcal{B}}) \leq P_b^{(u_q+2|\mathcal{T}^b|\varepsilon p_{\text{sum}})}(S_{\mathcal{B}})$ follows also from Proposition 5.5, thus $P_b^{(u_q)}(S^*) \leq P_b^{(u_q+2|\mathcal{T}^b|\varepsilon p_{\text{sum}})}(S) + \sum_{j \in U} p_j$,

which implies $P_a^{(u_q)}(S^*) \geq P_a^{(u_q+2|\mathcal{T}^b|\varepsilon p_{\text{sum}})}(S) - \sum_{j\in U} p_j$. Let $P_{S^*} := P_a^{(u_q)}(S^*)$ and $P_S := P_a^{(u_q+2|\mathcal{T}^b|\varepsilon p_{\text{sum}})}(S)$.

Note that $C_{\max} \leq u_q + 2|\mathcal{T}^b|\varepsilon p_{\text{sum}} + P_S/m + \varepsilon p_{\text{sum}}$ (Observation 5.8), $C^*_{\max} \geq u_q + P_{S^*}/m$ and $P_S \leq P_{S^*} + \sum_{j\in U} p_j$. From these, $C_{\max} \leq C^*_{\max} + 2|\mathcal{T}^b|\varepsilon p_{\text{sum}} + \sum_{j\in U} p_j/m + \varepsilon p_{\text{sum}}$ follows. Since $\sum_{j\in U} p_j \leq (|\mathcal{R}|+1)\cdot|\mathcal{T}^b|\varepsilon p_{\text{sum}}$ (Proposition 5.4), thus $C_{\max} \leq (1 + ((2|\mathcal{T}^b|+1)m + (|\mathcal{R}|+1)\cdot|\mathcal{T}^b|)\varepsilon)C^*_{\max}$, therefore we have proved Claim 3. □

The lemma follows from Claims 1, 2 and 3. □

Lemma 5.12. *For any fixed $\varepsilon > 0$, the running time of the algorithm is polynomial in the size of the input if $|\mathcal{T}^b|$ is a constant.*

Proof. Since the processing time of each big job is at least $\varepsilon p_{\text{sum}}$, the number of the big jobs is at most $\lfloor 1/\varepsilon \rfloor$, a constant, since ε is a constant by assumption. Thus, the total number of assignments of big jobs to time point in $\mathcal{T}^b$ is also constant. For each feasible assignment, a linear program of polynomial size in the input must be solved. This can be accomplished by the Ellipsoid method in polynomial time, see [5]. The remaining steps (rounding the solution, machine assignment and scheduling the small jobs) are obviously polynomial. □

Proof of Theorem 1.3. As we described above we can assume that the number of the distinct release dates is a constant, thus $|\mathcal{T}^b|$ is also a constant. The polynomial time complexity of the algorithm in the size of the input was shown in Lemma 5.12. According to Lemma 5.6, the performance ratio of the algorithm is $(1 + O(\varepsilon))$, where the constant factor c in $O(\cdot)$ does not depend on the input. Hence, to reach a desired performance ratio δ, we let $\varepsilon := \delta/c$, and perform the computations with this choice of ε. □

Remark 1. *Note that if a job is assigned to a v_ℓ, then $S_j \geq v_\ell$ at the end of the algorithm and each schedule such that this is true cannot hurt the resource constraint. Suppose that we fixed a big job assignment and solved the LP. Let*

- *If $j \in \mathcal{S}^a$, then let $\bar{r}_j := r_j$.*
- *If $j \in \mathcal{S}^b \cup \mathcal{B}$ and $\exists \ell : x_{j\ell} = 1$, then let $\bar{r}_j := v_\ell$.*
- *Otherwise let $\bar{r}_j := u_q$.*

After that, use the PTAS of [12] for the problem $P|\bar{r}_j|C_{\max}$. It is easy to prove that the schedule obtained is feasible and its makespan is at most $(1+\varepsilon)$ times the makespan of the schedule created by Algorithm A, thus it is also a PTAS for our problem.The algorithm of Hall and Shmoys works for arbitrary number of machines, however this number must be a constant when applied to our problem, otherwise the error bound breaks down.

6 $Pm|rm = const, q = const, r_j, ddc|C_{\max}$

Suppose that there is a dedicated machine for each job, or in other words, the assignment of jobs to machines is given in the input. As before, we can assume that the number of distinct release dates is a constant. Let M_{k_j} denote the machine on which we have to schedule J_j and $\mathcal{J}_k$ denote the set of jobs dedicated to M_k. We can model this problem with the IP (1)-(5) if we drop all the variables $x_{j\ell k}$ where $k \neq k_j$. Let us denote this new IP by (1')-(5'). We prove that there is a PTAS for this problem. The main idea of the algorithm is the same as in the previous section, however there are important differences.

Let $\varepsilon > 0$ be fixed, we divide the set of jobs into big and small ones ($\mathcal{B}$ and $\mathcal{S}$), and schedule them separately. These sets are the same as in Section 5. We assign the big jobs to time points in all possible ways (cf. Proposition 5.1). Notice that since $|\mathcal{B}| \leq 1/\varepsilon$, which is a constant because $\varepsilon > 0$ is fixed, the number of big job assignments is polynomial in the size of the input. We do the remaining part of the algorithm for each big job assignment. The first difference from the previous PTAS is the following: now we assign each small job in $\mathcal{S}^a$ to its release date and then we create the schedule S^1 from this partial assignment. Let $C^1_{\max}$ denote the makespan of S^1 and I_k denote the total idle time on machine k between u_q and $C^1_{\max}$ (if $C^1_{\max} \leq u_q$, then $I_k = 0$ for all $k \in \mathcal{M}$).

We need to schedule the small jobs in $\mathcal{S}^b$. We will schedule them in a suboptimal way and finally we choose the schedule with the lowest makespan. We will prove that the best solution found by the algorithm has a makespan of no more than $(1+\varepsilon)C^*_{\max}$ and the algorithm has a polynomial complexity.

For a fixed partial schedule we define the following linear program:

$$\min \bar{P} \tag{13}$$

s.t.

$$\sum_{j\in\mathcal{S}^b, v_\ell = u_q} p_j x^{small}_{j\ell k} \leq I_k + \bar{P}, \quad k \in \mathcal{M} \tag{14}$$

$$\sum_{k\in\mathcal{M}} \sum_{j\in\mathcal{S}^b} \sum_{\nu=1}^{\ell} a_{ij} x^{small}_{j\nu k} \leq \bar{b}_{\ell i}, \quad v_\ell \in \mathcal{T}^b,\ i \in \mathcal{R} \tag{15}$$

$$\sum_{j\in\mathcal{S}^b} p_j x^{small}_{j\ell k} \leq \max\{0, v_{\ell+1} - \bar{C}^{\mathcal{B}}_\ell(k)\} + \varepsilon p_{\text{sum}}, \quad v_\ell \in \mathcal{T}^b,\ k \in \mathcal{M} \tag{16}$$

$$\sum_{v_\ell \in \mathcal{T}} x^{small}_{j\ell k} = 1, \quad j \in \mathcal{S}^b \tag{17}$$

$$x^{small}_{j\ell k} = 0, \quad j \in \mathcal{S}^b,\ v_\ell \in \mathcal{T} \text{ such that } v_\ell < r_j, \text{ or } v_\ell > u_q \tag{18}$$

$$\bar{P} \geq 0 \tag{19}$$

$$x^{small}_{j\ell k} \geq 0, \quad j \in \mathcal{S}^b,\ v_\ell \in \mathcal{T}. \tag{20}$$

The variable $x^{small}_{j\ell k}$ exists only if J_j is dedicated to machine k, otherwise the notations are the same as before. Our objective ($\bar{P}$) is to minimize the increase of the

makespan compared to $C^1_{\max}$. The PTAS is as follows:

Algorithm B

1. Assign the big jobs to time points v_1 through v_τ which satisfies Proposition 5.1, and for each feasible assignment x^{big} do steps 2 - 7 :
2. Assign each small jobs in $\mathcal{S}^a$ to its release date. Create the partial schedule S^1 of the big jobs and the jobs in $\mathcal{S}^a$ and let $C^1_{\max}$ denote its makespan.
3. Define and solve linear program (13)-(20), and let $\bar{x}^{small}$ be an optimal basic solution.
4. Round each fractional value in $\bar{x}^{small}$ down to 0, and let $x^{small} := \lfloor \bar{x}^{small} \rfloor$ be the resulting partial assignment of small jobs, and $U \subset \mathcal{S}^b$ the set of fractional jobs in $\bar{x}^{small}$.
5. Create a (partial) schedule S^{part} of the jobs that we have already assigned to a time period. Let $C^{part}_{\max}$ denote the makespan of this schedule.
6. We need to schedule the fractionally assigned small jobs: do the following steps for every machine:
   i) Order the remaining jobs into an arbitrary order $(J_1, J_2, \ldots)$. We will schedule these jobs one by one:
   ii) Schedule J_j at the earliest idle time after u_q in the current schedule. If necessary, push to the right the later jobs and proceed with the next unscheduled job.
7. If the makespan of the complete schedule of all the jobs is better than the best solution found so far, then update the best solution.
8. After examining each feasible assignment of big jobs, output the best complete solution found.

Lemma 6.1. *Every complete solution (x^{big}, x^{small}) constructed by the algorithm is feasible for (1')-(5').*

Proof. (2') follows from (15) (the jobs scheduled after u_q cannot hurt this constraint), while the other constraints obviously met. □

Proposition 6.2. *In any basic solution of the linear program (7)-(12), there are at most $(|\mathcal{R}| + 1) \cdot |\mathcal{T}^b|$ fractional jobs.*

Proof. Similarly to Proposition 5.4. □

Proposition 6.3. *1. If a job J_j is assigned to v_ℓ at Step 1 or 2, then $S^{part}_j \leq S^1_j + \min\{\ell - 1, |\mathcal{T}^b|\}\varepsilon p_{\text{sum}}$.*

2. $C^{part}_{\max} \leq \max\{u_q, C^1_{\max}\} + \bar{P} + |\mathcal{T}^b|\varepsilon p_{\text{sum}}$.

Proof. Similarly to Proposition 5.5. □

Lemma 6.4. *The algorithm constructs at least one complete assignment* (x^{big}, x^{small}) *whose value is at most* $(1 + O(\varepsilon))$ *times the optimum makespan* $C^*_{\max}$.

Proof. Consider an optimal schedule $S^* = (\hat{x}^{big}, \hat{x}^{small})$ of (1')-(5') that satisfy the condition of Proposition 5.1. The algorithm will examine $\hat{x}^{big}$, since it is a feasible big job assignment. The partial assignment of the small jobs in $\mathcal{S}^b$ in S^* determines a feasible solution of (13)-(20), thus $\max\{u_q, C^1_{\max}\} + \bar{P} \leq C^*_{\max}$.

According to Proposition 6.3 $C^{part}_{\max} \leq \max\{u_q, C^1_{\max}\} + \bar{P} + |\mathcal{T}^b|\varepsilon p_{\text{sum}}$, and $C_{\max} \leq C^{part}_{\max} + (|\mathcal{R}| + 1) \cdot |\mathcal{T}^b|\varepsilon p_{\text{sum}}$ follows from Proposition 6.2. Therefore $C_{\max} \leq (1 + ((|\mathcal{R}| + 2) \cdot |\mathcal{T}^b|)m\varepsilon)C^*_{\max}$. □

Lemma 6.5. *For any fixed* $\varepsilon > 0$*, the running time of the algorithm is polynomial in the size of the input.*

Proof. Similarly to Lemma 5.12. □

Theorem 6.6. *There is a PTAS for the problem* $P|ddc, m = const., rm = const., q = const., \#\{r_j : r_j < u_q\} = const.|C_{\max}$.

Proof. Follows from Lemmas 6.4 and 6.5. □

Remark 2. *Suppose that, there is a dedicated machine for each job in a given set* $\mathcal{J}' \subset \mathcal{J}$ *and we can schedule each job in* $\mathcal{J} \setminus \mathcal{J}'$ *on any machine. We still have a PTAS for this case: the main difference is that at Step 6 we first have to schedule the jobs in* $\mathcal{J}'$ *and then the remaining jobs similarly to Step 6 in Algorithm A.*

7 $Pm|rm = 1, p_j = a_j, r_j|C_{\max}$

Suppose that we have only one resource and $p_j = a_j$ for each $j \in \mathcal{J}$. We prove that there is a PTAS for this problem even if the number of the supplies is part of the input and the jobs have release dates. Briefly, the algorithm is the following: consider each big job assignment in turn, and for each of them we guess approximately the makespan by a logarithmic search, and try to schedule the small jobs within the guessed makespan. Let $p^{\mathcal{S}}_{\text{sum}}$ denote the total processing time of the small jobs. Formally, the algorithm is as follows:

Algorithm C

1. Assign the big jobs to time points v_1 through v_τ and to machines 1 to $|\mathcal{M}|$ in all possible ways which satisfy Proposition 5.1, and for each feasible assignment x^{big} do steps 2 - 4 :

2. Create a partial schedule S^{part} of the big jobs. Let $C^{part}_{\max}$ denote the makespan of this schedule.

3. Let $C^-_{\max} := \max\{v_\tau, C^{part}_{\max}\}$ and $C^+_{\max} := C^-_{\max} + p^{\mathcal{S}}_{\text{sum}}/m + 2\varepsilon p_{\text{sum}}$. Do logarithmic search for $C_{\max}$:

   i) If $C^-_{\max} \geq C^+_{\max}$, then return with the best schedule found so far.

   ii) Let $\bar{C}_{\max} := \lfloor (C^-_{\max} + C^+_{\max})/2 \rfloor$. Invoke Algorithm C2 with the partial schedule and $\bar{C}_{\max}$.

   iii) If the output of Algorithm C2 is a complete schedule, then let $C^+_{\max} := \bar{C}_{\max}$ and go to Step 3i.

   iv) If the output is 'NO', then let $C^-_{\max} := \bar{C}_{\max} + 1$ and go to Step 3i.

4. If the makespan of the complete schedule of all the jobs is better than the best solution found so far, then update the best solution.

5. After examining each feasible assignment of big jobs, output the best complete solution found.

Algorithm C2

Input: a partial schedule and $\bar{C}_{\max}$.
Output: a feasible complete schedule (that 'contains' the partial schedule) with a makespan of $\bar{C}_{\max}$ or 'NO'.

1. If $C^{part}_{\max} > \bar{C}_{\max}$, then stop with the answer 'NO'. Otherwise, put the remaining (small) jobs into an ordered list in a non-increasing release date order: $J_1, J_2, \ldots$. We will schedule these jobs as follows:

2. Let $C_k := \bar{C}_{\max}$ for all $k \in \mathcal{M}$. For $\ell = \tau, \tau - 1, \ldots, 1$ do Step 3.

3. For $k = 1, \ldots, m$ do the following steps:

   i) Let t be the maximal index such that the total processing time of the first t jobs from the ordered list is at most $C_k - \bar{C}^{\mathcal{B}}_\ell(k)$.

   ii) Schedule the first t jobs on machine k before C_k without idle time and delete them from the ordered list. Push to the right the big jobs assigned to v_ℓ so that there will be no idle time after them. If this new partial schedule hurts (2) or (4) then return with 'NO'. Otherwise, let C_k be the starting time of the first job that starts after v_ℓ on machine k.

4. If we have scheduled every job, then return with this complete schedule. Otherwise return with 'NO'.

See Figure 2 for an illustration.

Remark 3. *It is enough to consider only the integer solutions, since by delaying some jobs less than one can cause at most* $O(\varepsilon)C^*_{\max}$ *increase in the objective function: we have assumed (Assumption 2) that* $\varepsilon > 1/n$*, thus* $\varepsilon > 1/(mC^*_{\max})$*, therefore* $C^*_{\max} + 1 = C^*_{\max}(1 + 1/C^*_{\max}) < C^*_{\max}(1 + m\varepsilon)$.

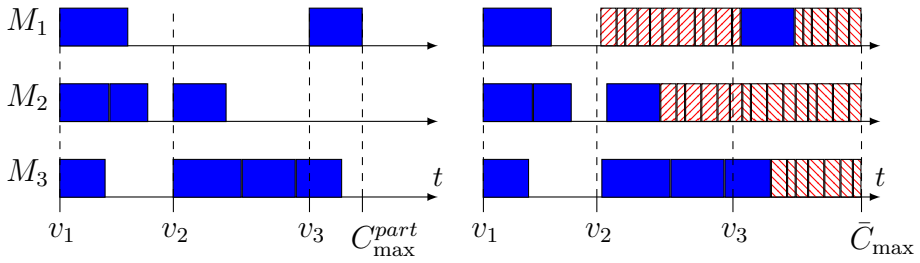

Figure 2: A partial schedule on the left and the schedule was created by Algorithm C2 after $\ell = 2$ on the right.

Lemma 7.1. *Every complete schedule constructed by Algorithm C is feasible for (1)-(5).*

Proof. Algorithm C only examines feasible big job assignments, while Algorithm C2 does not give a schedule that hurts (2), thus every complete schedule is feasible for (2). Any complete schedule found is obviously feasible for the other constraints. □

Observation 7.2. For every $k \in \mathcal{M}$, if there is at least one unscheduled small job after Algorithm C2 updates C_k in some iteration ℓ, then $\max\{\bar{C}^{\mathcal{B}}_{\ell-1}(k), v_\ell\} \leq C_k \leq \max\{\bar{C}^{\mathcal{B}}_{\ell-1}(k), v_\ell\} + \varepsilon p_{\text{sum}}$.

Proposition 7.3. *If $\bar{C}_{\max} \geq C^*_{\max} + \varepsilon p_{\text{sum}}$ and the partial schedule S^{part} is part of an optimal schedule (i.e. every big job is assigned to the same v_ℓ in S^* and S^{part}), then Algorithm C2 returns a feasible schedule of makespan $\bar{C}_{\max}$.*

Proof. Let S^* be an optimal schedule. Since $\bar{C}_{\max} \geq C^*_{\max} \geq C^{part}_{\max}$, thus the algorithm does not stop at Step 1. For every $\ell = q, q-1, \ldots, 1$, the algorithm can do three things at Step 3: (i) creates a schedule without any idle time between C_k and $\bar{C}_{\max}$ ($\forall k \in \mathcal{M}$); (ii) schedules every small job after v_ℓ; or (iii) returns with 'NO'.

Claim 4. *For every ℓ, the total resource requirement (processing time) of the small jobs starting after v_ℓ in S^* is at most the total resource requirement of the small jobs scheduled after v_ℓ by the algorithm.*

Proof. If every small job is scheduled by the algorithm after v_ℓ, then the claim is trivial. Otherwise each machine is working between C_k and $\bar{C}_{\max}$. Since $\bar{C}_{\max} \geq C^*_{\max} + \varepsilon p_{\text{sum}}$, the statement follows from Observation 7.2. □

Therefore the algorithm cannot hurt (2) or (4) at Step 3. Applying Claim 4 with $\ell = 1$ we get that every small job is scheduled in this case, thus the proposition follows. □

Proposition 7.4. *If the partial schedule S^{part} is part of an optimal schedule, then $C^*_{\max} \leq \max\{v_\tau, C^{part}_{\max}\} + p^{\mathcal{S}}_{\text{sum}}/m + \varepsilon p_{\text{sum}}$.*

Proof. Create schedule S' from S^{part} as follows: put the small jobs in arbitrary order, and schedule them one by one at the earliest possible time moment after $\max\{v_\tau, C^{part}_{\max}\}$ on any machine. Since the difference between the finishing time of two arbitrary machines is at most $\varepsilon p_{\text{sum}}$, then the makespan of S' is at most $\max\{v_\tau, C^{part}_{\max}\} + p^{\mathcal{S}}_{\text{sum}}/m + \varepsilon p_{\text{sum}}$. Since S' is feasible, thus the proposition follows. □

Lemma 7.5. *Algorithm C constructs at least one complete schedule whose makespan is at most* $(1 + O(\varepsilon))C^*_{\max}$.

Proof. Let S^* be an optimal schedule that satisfies the condition of Proposition 5.1. The algorithm will examine its big job assignment since it is feasible. If $\bar{C}_{\max} \geq C^*_{\max} + \varepsilon p_{\text{sum}}$, then Algorithm C2 creates a feasible schedule (Proposition 7.3) with a makespan of $\bar{C}_{\max}$. We claim that Algorithm C invokes Algorithm C2 with $\bar{C}_{\max} = C^*_{\max} + \lceil \varepsilon p_{\text{sum}} \rceil = (1 + O(\varepsilon))C^*_{\max}$ (since $p_{\text{sum}} \leq mC^*_{\max}$). However this claim follows from Proposition 7.4 since $\max\{v_\tau, C^{part}_{\max}\} \leq C^*_{\max} \leq \max\{v_\tau, C^{part}_{\max}\} + p^{\mathcal{S}}_{\text{sum}}/m + \varepsilon p_{\text{sum}}$, thus $C^-_{\max} \leq C^*_{\max} + \lceil \varepsilon p_{\text{sum}} \rceil \leq C^+_{\max}$. □

Lemma 7.6. *For any fixed* $\varepsilon > 0$*, the running time of Algorithm C is polynomial in the size of the input.*

Proof. The number of the big jobs is at most $1/\varepsilon$, thus the number of the big job assignments is polynomial. For a fixed big job assignment the algorithm invokes Algorithm C2 at most $\lceil \log(p^{\mathcal{S}}_{\text{sum}}/m + \varepsilon p_{\text{sum}}) \rceil$ times. The running time of Algorithm C2 is at most $O(n \log n + qmn)$. □

Theorem 7.7. *There is a PTAS for the problem* $Pm|rm = 1, p_j = a_j|C_{\max}$.

Proof. Follows from Lemmas 7.1, 7.5 and 7.6. □

8 $P|rm = 1, p_j = a_j, r_j, ddc|C_{\max}$

In this section we sketch a PTAS for the problem of Section 7 in case of dedicated machines. We only have to modify the Algorithm C and C2 at some steps:

- At Step 1 in Algorithm C, we do not have to assign the big jobs to machines. At Step 3, let $C^+_{\max} := \max_{k \in \mathcal{M}}\{C^{part}_{\max}(k) + \sum_{j \in \mathcal{S}_k} p_j\} + \varepsilon p_{\text{sum}}$, where $C^{part}_{\max}(k)$ is the finishing time of machine k in S^{part} and $\mathcal{S}_k := \mathcal{J}_k \cap \mathcal{S}$.
- At Step 1 in Algorithm C2, we create m lists, the list k contains the small jobs dedicated to machine k. At Step 3 we always choose (and delete) jobs from the appropriate list.

The proofs are similar to those in Section 7. We can generalize this result in case of partly dedicated jobs as before.

Appendix

Proof of Proposition 5.1. Let $\mathcal{J}^a(\hat{x})$ be the subset of jobs with $\hat{x}_{j\ell k} = 1$ for some $v_\ell > u_q$ and $k \in \mathcal{M}$. We define a new solution $\tilde{x}$ in which those jobs in $\mathcal{J}^a(\hat{x})$ are reassigned to new time points (but to the same machine) and show that $C_{\max}(\tilde{x}) \leq C_{\max}(\hat{x})$. Let $\tilde{x} \in \{0,1\}^{\mathcal{J}\times\mathcal{T}\times\mathcal{M}}$ be a binary vector which agrees with $\hat{x}$ for those jobs in $\mathcal{J} \setminus \mathcal{J}^a(\hat{x})$. For each $j \in \mathcal{J}^a(\hat{x})$, let $\tilde{x}_{j\ell k} = 1$ for $v_\ell = \max\{u_q, r_j\}$ and for a k such that $\exists \ell' : \hat{x}_{j\ell' k} = 1$, and 0 otherwise. We claim that $\tilde{x}$ is a feasible solution of (1)-(5), and that $C_{\max}(\tilde{x}) \leq C_{\max}(\hat{x})$. Feasibility of $\tilde{x}$ follows from the fact that u_q is the last time point when some resource is supplied, and that no job is assigned to some time point before its release date. As for the second claim, consider the objective function (1). We will verify that for each $k \in \mathcal{M}$ and $\ell = 1, \ldots, \tau$,

$$v_\ell + \sum_{j\in\mathcal{J}} \sum_{\nu=\ell}^{\tau} p_j \tilde{x}_{j\nu k} \leq v_\ell + \sum_{j\in\mathcal{J}} \sum_{\nu=\ell}^{\tau} p_j \hat{x}_{j\nu k}, \tag{21}$$

from which the claim follows. If $v_\ell \leq u_q$, the left and the right-hand sides in (21) are equal. Now consider any ℓ with $v_\ell > u_q$. Since no job in $\mathcal{J}^a(\hat{x})$ is assigned to a later time point in $\tilde{x}$ than in $\hat{x}$, the inequality (21) is verified again. □

Proof of Proposition 5.2. Suppose $(\hat{x}^{big}, \hat{x}^{small})$ is an optimal solution which does not meet the property claimed. Without loss of generality, we may assume that in the optimal schedule corresponding to $(\hat{x}^{big}, \hat{x}^{small})$, for each $v_k \in \mathcal{T}$, small jobs assigned to v_k follow the big ones assigned to v_k. Let $v_\ell \in \mathcal{T}^b$ be the smallest time point for which (6) is violated. Then some small jobs assigned to v_ℓ necessarily start after $v_{\ell+1}$ in any schedule corresponding to $(\hat{x}^{big}, \hat{x}^{small})$. Since all small jobs are of processing time less than $\varepsilon p_{\text{sum}}$, we can reassign some of the small jobs from time point v_ℓ to $v_{\ell+1}$ until (6) is satisfied for v_ℓ. Clearly, such a reassignment of small jobs does not increase the length of the schedule. Then we proceed with the next time point in $\mathcal{T}$ until we get a schedule meeting (6). □

Acknowledgments

This work has been supported by the OTKA grant K112881. The research of Tamás Kis has been supported by the János Bolyai research grant BO/00412/12/3 of the Hungarian Academy of Sciences.

References

[1] D. Briskorn, B.-C. Choi, K. Lee, J. Leung, and M. Pinedo. Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, 207:605–619, 2010.

[2] D. Briskorn, F. Jaehn, and E. Pesch. Exact algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 16:105–115, 2013.

[3] J. Carlier. *Problèmes d'ordonnancements à contraintes de ressources: algorithmes et complexité. Thèse d'état.* Université Paris 6, 1984.

[4] J. Carlier and A. H. G. Rinnooy Kan. Scheduling subject to nonrenewable resource constraints. *Operational Research Letters*, 1:52–55, 1982.

[5] P. Gács and L. Lovász. Khachiyan's algorithm for linear programming. *Mathematical Programming Studies*, 14:61–81, 1981.

[6] E. R. Gafarov, A. A. Lazarev, and F. Werner. Single machine scheduling problems with financial resource constraints: Some complexity results and properties. *Mathematical Social Sciences*, 62:7–13, 2011.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, LA: Freeman, 1979.

[8] A. Grigoriev, M. Holthuijsen, and J. van de Klundert. Basic scheduling problems with raw material constraints. *Naval Research of Logistics*, 52:527–553, 2005.

[9] P. Györgyi and T. Kis. Approximation schemes for single machine scheduling with non-renewable resource constraints. *Journal of Scheduling*, 17:135–144, 2014.

[10] P. Györgyi and T. Kis. Approximability of scheduling problems with resource consuming jobs. *Annals of Operations Research*, pages –, 2015.

[11] P. Györgyi and T. Kis. Reductions between scheduling problems with non-renewable resources and knapsack problems. *Theoretical Computer Science*, 565:63–76, 2015.

[12] L. Hall and D. B. Shmoys. Approximation schemes for constrained scheduling problems. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 134–139. IEEE, 1989.

[13] P. Laborie. Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143:151–188, 2003.

[14] E. Morsy and E. Pesch. Approximation algorithms for inventory constrained scheduling on a single machine on the same problem. *Journal of Scheduling*, pages –, 2015.

[15] K. Neumann and C. Schwindt. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56:513–533, 2003.

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

[17] A. Toker, S. Kondakci, and N. Erkip. Scheduling under a non-renewable resource constraint. *Journal of the Operational Research Society*, 42:811–814, 1991.