A PTAS for a resource scheduling problem with arbitrary number of parallel machines

Péter Györgyi[a,b]

[a] *Department of Operations Research, Loránd Eötvös University, H1117 Budapest, Pázmány Péter sétány 1/C, Hungary*
[b] *Institute for Computer Science and Control, H1111 Budapest, Kende str. 13–17, Hungary*

Abstract

In this paper we study a parallel machine scheduling problem with non-renewable resource constraints. That is, besides the jobs and machines, there is a common non-renewable resource consumed by the jobs, which has an initial stock and some additional supplies over time. Unlike in most previous results, the number of machines is part of the input. We describe a polynomial time approximation scheme for minimizing the makespan.

Keywords: parallel machine scheduling, non-renewable resource, approximation scheme

1. Introduction

In this paper we study a parallel machine scheduling problem and describe a polynomial time approximation scheme (PTAS) for it. In our problem, the jobs have an additional resource requirement: there is a non-renewable resource (like raw material, energy, or money) consumed by the jobs. The resource has an initial stock, which is replenished at some a-priori known moments of time. As usual, each job can be scheduled on any machine, the job processing times do not depend on the machines assigned, machines can perform only one job at a time, and preemption of jobs is not allowed. The objective is to minimize the maximal job-completion time, 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 common resource consumed by some, or possibly all of the jobs. Each job J_j has a processing time $p_j \in \mathbb{Z}_+$ and a resource requirement $a_j \in \mathbb{Z}_{\geq 0}$ from the common resource, noting that $a_j = 0$ is possible. The resource is supplied in q different time moments, $0 = u_1 < u_2 < \ldots < u_q$; the number $\tilde{b}_\ell \in \mathbb{Z}_+$ represents the quantity supplied at u_ℓ, $\ell = 1, 2, \ldots, q$. A *schedule* σ specifies a machine and the starting time S_j for each job, and it is *feasible* if (i) on every machine the jobs do not overlap in time, and if (ii) at any time point t the total material supply from the resource is at least the total request of those jobs starting not later than t, i.e., $\sum_{(\ell \,:\, u_\ell \leq t)} \tilde{b}_\ell \geq \sum_{(j \,:\, S_j \leq t)} a_j$. The objective is to minimize the makespan, i.e., the completion time of the job finished last.

This problem is a sub-problem of a more general resource scheduling problem: in the general case there are r resources, the requirements a_j and the supplies b_ℓ are r-dimensional vectors. We denote our problem by $P|rm = 1|C_{\max}$, where $rm = 1$ indicates that there is only one single non-renewable resource. Since the makespan minimization problem with resource consuming jobs on a single machine is NP-hard even if there are only two supply dates [2], the studied problem is 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 real-world applications (e.g. [1], [4]).

1.1. Main result and structure of the paper

Section 2 summarizes the previous results, while Section 3 simplifies the resource scheduling problem with some observations and gives an integer programming model of the problem. In Section 4, we prove the following:

Theorem 1. *There is a PTAS for* $P|rm = 1|C_{\max}$.

There are several approximation schemes for similar scheduling problems with non-renewable resource constraints (see Section 2), however, to our best knowledge, this is the first time, that an arbitrary number of parallel machine is considered in an approximation algorithm for scheduling with non-renewable resources. Note that the latter problem is already APX-hard in case of two resources ([11]), so limiting the number of resources to one is necessary to have a PTAS unless P = NP. The problem $P|rm = 1|C_{\max}$ was the only problem with unknown approximabilty status in the class $P|rm|C_{\max}$ ([11]).

Our PTAS reuses ideas from known PTAS-es designed for $P||C_{\max}$ (e.g. [13],[12]). Actually, we invoke a variant of the latter. However, there are no resource constraints in

Email address: gyorgyi.peter@sztaki.mta.hu (Péter Györgyi)

those PTAS-es, therefore the jobs differ only in their processing times. Rounding techniques are useful in a PTAS to simplify the instances (e.g. Lemmas 1 and 2), because they introduce only small errors, but rounding the resource supplies or resource requirements does not seem a viable approach. Instead, we will sort the jobs into different categories, and use enumeration to find suboptimal schedules for the problem with rounded processing times.

1.2. Terminology

An *optimization problem* Π consists of a set of instances, where each instance has a set of *feasible solutions*, and each solution has an (objective function) value. In a *minimization problem* a feasible solution of minimum value is sought, while in a maximization problem one of maximum value. 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

Makespan minimization on parallel machines is one of the oldest problem of scheduling theory. The problem is strongly NP-hard ([6]), but there is a PTAS for it ([13]).

Scheduling problems with resource consuming jobs were introduced by [2], [3], and [15]. In [2], the computational complexity of several variants with a single machine was established, while in [3] activity networks requiring only non-renewable resources were considered. In [15] a parallel machine problem with preemptive jobs was studied with a single non-renewable resource. This 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 a strong contrast to the NP-hardness of the scheduling problem analyzed in this paper. Further results can be found in e.g., [16], [17], [7], [5], [8], [9], [10], [14], [11].

In [8], [9] and [10] there are several approximability results for the single machine variant of the problem. [11] provided PTAS-es for some parallel machine variant of the problem and showed that the problem with two resources and two supplies is APX-hard. See also [11] for further previous results of the topic.

3. Preliminaries

Note that the following assumption holds without loss of generality and it has two easy corollaries:

Assumption 1. $\sum_{\ell=1}^{q} \tilde{b}_\ell = \sum_{j\in\mathcal{J}} a_j$.

Corollary 1. *$C^*_{\max} > u_q$ and we have enough resource for each job that starts after u_q.*

Observation 1. *For a PTAS, it is sufficient to provide a schedule with a makespan of $(1+c\varepsilon)$ times the optimum value, where c is a constant i.e. it does not depend on the input. Hence, to reach a desired performance ratio δ, we let $\varepsilon := \delta/c$, and perform the computations with the choice of ε.*

The observation above shows the meaning of the next lemmas.

Lemma 1. *With $1+\varepsilon$ loss, we can assume that all processing times are integer powers of $1+\varepsilon$. (trivial)*

Lemma 2. *([11]) In order to have a PTAS for $P|rm|C_{\max}$, it suffices to provide a family of algorithms $\{A_\varepsilon\}_{\varepsilon>0}$ such that A_ε is an ε-approximation algorithm for the restricted problem where the supply dates before u_q are from the set $\{\ell\varepsilon u_q : \ell = 0,1,2,\ldots,\lfloor 1/\varepsilon\rfloor\}$.*

We can model $P|rm=1|C_{\max}$ with a mathematical program with integer variables in a way similar to that of [11]. We define the values $b_\ell := \sum_{\nu\,:\,u_\nu\le u_\ell} \tilde{b}_\nu$, that is, b_ℓ equals the total amount supplied from the resource up to u_ℓ and let $\mathcal{T} := \{u_1, u_2, \ldots, u_q\}$. We introduce $q\cdot|\mathcal{J}||\mathcal{M}|$ binary decision variables $x_{j\ell k}$, $(j\in\mathcal{J}, \ell = 1,\ldots,q, k\in\mathcal{M})$ such that $x_{j\ell k}=1$ if and only if job j is assigned to machine k and to the time point u_ℓ, which means that the requirements of job j must be satisfied by the resource supplies up to time point u_ℓ. The mathematical program is

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

s.t.

$$\sum_{k\in\mathcal{M}} \sum_{j\in\mathcal{J}} \sum_{\nu=1}^{\ell} a_j x_{j\nu k} \le b_\ell, \quad u_\ell\in\mathcal{T} \tag{2}$$

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

$$x_{j\ell k}\in\{0,1\}, \quad j\in\mathcal{J},\ u_\ell\in\mathcal{T},\ k\in\mathcal{M}. \tag{4}$$

The objective function expresses the completion time of the job finished last using the observation that for every machine there is a time point from which the machine processes the jobs without idle times. Constraints (2) ensure that the jobs assigned to time points u_1 through u_ℓ use only the resources supplied up to time u_ℓ. Equations (3) ensure that all jobs are assigned to some machine and

time point. 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 u_ℓ.

$\bar{x}$ is a *partial* assignment, if it satisfies (4) and $\sum_{k\in\mathcal{M}}\sum_{\ell=1}^{q} x_{j\ell k} \leq 1, \quad j \in \mathcal{J}$. If it satisfies also (2), then it is a *feasible* partial assignment.

Subroutine Sch describes how we create a (partial) schedule from a (partial) assignment.

Subroutine Sch ([11])

Input: $\bar{\mathcal{J}} \subseteq \mathcal{J}$ and $\bar{x}$ such that for each $j \in \bar{\mathcal{J}}$ there exists a unique (ℓ, k) with $\bar{x}_{j\ell k} = 1$, and $\bar{x}_{j\ell k} = 0$ otherwise.

Output: partial schedule S^{part} of the jobs in $\bar{\mathcal{J}}$.

1. S^{part} is initially empty, then we schedule the jobs on each machine in increasing u_ℓ order (first we schedule those jobs assigned to u_1, and then those assigned to u_2, etc.):
2. When scheduling the next job with $\bar{x}_{j\ell k} = 1$, then it is scheduled at time $\max\{u_\ell, C_{last}(k)\}$, where $C_{last}(k)$ is the completion time of the last job scheduled on machine M_k, or 0 if no job has been scheduled yet on M_k.

Remark 1. *Note that if $\bar{x}$ is feasible partial assignment, then S^{part} is a feasible partial schedule, since:*

- *the jobs on the same machine cannot overlap in time,*
- *$S_j \geq u_\ell$ holds for each job assigned to u_ℓ, thus (2) ensures that we have enough resource to schedule the jobs.*

Suppose we have a partial schedule S^{part} and consider an idle period I on some machine M_k. Suppose j_1 is not scheduled in S^{part}, and we schedule j_1 on M_k with starting time $t_1 \in I$. This transforms S^{part} as follows. For each job j scheduled on M_k in S^{part} with $S_j^{part} > t_1$, let $P_k[t_1, S_j^{part}]$ denote the total processing time of those jobs scheduled on M_k in S^{part} between t_1 and S_j^{part}. We update the start-time of j to $\max\{S_j^{part}, t_1 + p_{j_1} + P_k[t_1, S_j^{part}]\}$. The start time of any other job does not change. Note that the value of $P_k[t_1, S_j^{part}]$ can be computed before the transformation, thus the new starting times do not depend on the order of updating. See Figure 1 for an illustration.

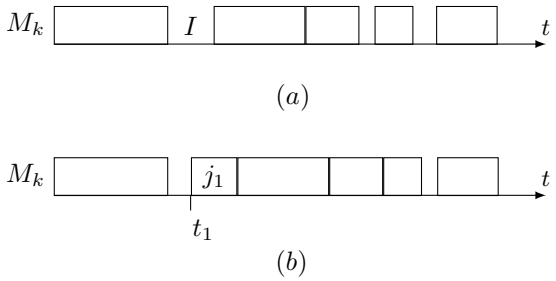

Figure 1: Inserting job (j_1) into a partial schedule with $S_{j_1} = t_1$. (a): the original partial schedule on M_k. (b): the new schedule on M_k after inserting j_1.

Claim 1. *The jobs do not overlap in time in the resulting schedule.*

Proof. Follows from the definition of $P_k[t_1, S_j^{part}]$. □

4. A PTAS for the problem $P|rm = 1|C_{\max}$

In this section we prove Theorem 1. Let $\varepsilon > 0$ be fixed. It is enough to deal with the case where $q = \lceil 1/\varepsilon \rceil + 1$ and the supply dates before u_q are from the set $\{\ell\varepsilon u_q : \ell = 0, 1, 2, \ldots, \lfloor 1/\varepsilon \rfloor\}$ (Lemma 2) and according to Lemma 1, it is enough to provide a PTAS for the problem instances, where each processing time is an integer power of $1 + \varepsilon$.

For the PTAS it is useful to divide $\mathcal{J}$ into three subsets. A job j is *small* ($j \in \mathcal{S}$), if $p_j \leq \varepsilon^2 u_q$, it is *big* ($j \in \mathcal{B}$), if $\varepsilon^2 u_q < p_j < (1/\varepsilon)u_q$ and it is *huge* ($j \in \mathcal{H}$), if $p_j \geq (1/\varepsilon)u_q$. We can assume that each huge job starts after u_q since in this case a delay of u_q is at most an ε fraction of the makespan (see Observation 1).

This partition has several advantages: we do not have to deal with the resource requirements of the huge jobs (cf. Corollary 1), there are a constant number of possible values for the processing time of the big jobs (see the next paragraph) and even $O(1/\varepsilon)$ badly scheduled small jobs cause only $O(\varepsilon u_q)$ delay which is $O(\varepsilon C^*_{\max})$ since $C^*_{\max} > u_q$.

Now we determine the number of the possible distinct processing times of the big jobs: we have to count the number of the different δ values such that $\delta \in \mathbb{Z}$ and $\varepsilon^2 u_q < (1+\varepsilon)^\delta < (1/\varepsilon)u_q$. Since $\varepsilon^2 u_q \cdot (1+\varepsilon)^{3\log_{1+\varepsilon}(1/\varepsilon)} = (1/\varepsilon)u_q$, there are $k_1 := \lfloor 3\log_{1+\varepsilon}(1/\varepsilon) \rfloor$ possible values. Note that k_1 is a constant. Let δ_1 denote the smallest number such that $(1+\varepsilon)^{\delta_1} > \varepsilon^2 u_q$ and $\mathcal{B}_1$ the set of big jobs with processing time $(1+\varepsilon)^{\delta_1}$. For $p = 2, \ldots, k_1$, let $\mathcal{B}_p$ denote the set of big jobs with processing time $(1+\varepsilon)^{\delta_1+p-1}$.

Before going into details we would like to present the main ideas of the PTAS. After u_q we do not have to deal with the resource constraints (Corollary 1) and this is a great relief: we will use a modified version of an algorithm devised for $P||L_{\max}$ to schedule the jobs that we have not scheduled earlier. However, to determine the first part of the schedule we need a more sophisticated method: the main idea is a tricky enumeration method with a number of technical details. For each pair (M_k, u_ℓ), where $k \in \{1, \ldots, m\}$ and $\ell \in \{1, \ldots, q-1\}$, we guess the number of the big jobs from each type $\mathcal{B}_p$ ($p \in 1, \ldots, k_1$) and approximately the total processing time of the small jobs that start in $[u_\ell, u_{\ell+1})$ on M_k. For this purpose we examine a lot of cases and finally we choose the best one. In the next paragraphs we present the details of the method and show that the number of the possible guesses is polynomial.

A guess will describe an *assignment* A for each machine and each assignment consists of $(q-1)(k_1+1)$ numbers: for each $\ell < q$ it contains $\Gamma_\ell = (\gamma_{\ell,1}, \gamma_{\ell,2}, \ldots, \gamma_{\ell,k_1}, g_\ell)$, where each coordinate is from the set $\{0, 1, \ldots, \lceil 1/\varepsilon \rceil + 1\}$. $\gamma_{\ell,p}$ describes the number of the big jobs from $\mathcal{B}_p$ that start in $[u_\ell, u_{\ell+1})$ and we guess the total processing time of the

small jobs that start in $[u_\ell, u_{\ell+1})$ in the form of $g_\ell \cdot (\varepsilon^2 u_q)$.

From the definition of the assignment we get the number of the different assignments is at most $k_2 := (1/\varepsilon + 2)^{(q-1)\cdot(k_1+1)}$. Note that k_2 is a constant and for any machine scheduled using this assignment, the number of big jobs that start in $[u_\ell, u_{\ell+1})$ ($\ell \in \{1, \ldots, q-1\}$) is at most $\lceil 1/\varepsilon \rceil$ (cf. the condition on u_ℓ in Lemma 2 and the definition of the big jobs) and the total processing time of the small jobs that start in $[u_\ell, u_{\ell+1})$ is at most $(\varepsilon + \varepsilon^2)u_q$ (since $u_{\ell+1} - u_\ell \leq \varepsilon u_q$ and the last job must finish before $u_{\ell+1} + \varepsilon^2 u_q$). Hence, the guesses describe each possible big job layout and give an approximation to the total processing time of the small jobs in $[u_\ell, u_{\ell+1})$.

A tuple $T := (t_1, t_2, \ldots, t_{k_2})$ describes the number of the machines that uses the different assignments $A_1, \ldots, A_{k_2}$. Note that the number of these tuples is at most $\binom{m+k_2}{k_2}$, thus it is polynomial (details in Proposition 2). We examine all tuples and either create a schedule according to the tuple or declare that the tuple is unfeasible.

Example 1. *Suppose that we have 3 machines, $q = 4$ and $k_1 = 2$. If $T = (1, 2, 0, 0, \ldots)$, $A_i = (\Gamma_1^i, \Gamma_2^i, \Gamma_3^i)$ ($i = 1, \ldots, k_2$), and $\Gamma_\ell^i = (\gamma_{\ell,1}^i, \gamma_{\ell,2}^i, g_\ell^i)$, then it means the following: we have one machine that uses the assignment A_1 and two that use A_2. Γ_ℓ^i describes that we want to assign (i) $\gamma_{\ell,p}^i$ big jobs from $\mathcal{B}_p$ ($p = 1, 2$), (ii) small jobs with a total processing time of roughly $g_\ell^i \cdot (\varepsilon^2 u_q)$ to u_ℓ on each machine with the assignment A_i.*

We will use a greedy algorithm to define the assignment of the small jobs according to the guess. We create a (partial) schedule from the (partial) assignment and after that, we invoke another algorithm to schedule the remaining jobs. The main algorithm is as follows:

Algorithm A

Initialization: S^{best} is a schedule where each job is scheduled on M_1 after u_q.

1. For each tuple $T = (t_1, \ldots, t_{k_2})$, do Steps 2–5:
2. Invoke Algorithm Assign to create an assignment $\hat{x}$ of the jobs from T. If this assignment violates at least one of the constraints in (2), then proceed with the next tuple.
3. Create a partial schedule S^{part} from $\hat{x}$ with Subroutine Sch. Let $C_{\max}^{part}(k)$ be the time when M_k finishes S^{part}.
4. Invoke the algorithm of the Appendix with $\max\{C_{\max}^{part}(k), u_q\}$ amount of preassigned work on M_k ($k = 1, 2, \ldots, m$) to schedule the remaining jobs. Let S^{act} be the resulting schedule.
5. If $C_{\max}(S^{act}) < C_{\max}(S^{best})$, then let $S^{best} := S^{act}$.
6. After examining each feasible assignment before u_q, output S^{best}.

The next algorithm assigns big and small jobs to pairs (M_k, u_ℓ), where $\ell < q$ with respect to a tuple $T = (t_1, \ldots, t_{k_2})$. Machines $M_1, \ldots, M_{t_1}$ will get jobs with respect to assignment A_1, $M_{t_1+1}, \ldots, M_{t_1+t_2}$ with respect to A_2, etc.

Algorithm Assign

Input: a tuple T. Output: a (partial) assignment $\hat{x}$.

1. For each $p = 1, \ldots, k_1$, order the big jobs from $\mathcal{B}_p$ in non-decreasing a_j order (lists L_p) and let L be the list of small jobs in non-increasing p_j/a_j order.
2. Assign big and small jobs to machine-supply date pairs (M_k, u_ℓ) in the order (M_1, u_1), (M_2, u_1), $\ldots, (M_m, u_1)$, (M_1, u_2), $\ldots$, (M_m, u_2), (M_1, u_3), $\ldots$, (M_m, u_{q-1}). Suppose the next pair is (M_k, u_ℓ), and the assignment of M_k is $A_i = (\Gamma_\ell)_{\ell=1}^{q-1}$, where $\Gamma_\ell = (\gamma_{\ell,1}, \ldots, \gamma_{\ell,k_1}, g_\ell)$,

   Assignment of big jobs: for each $p = 1, \ldots, k_1$, assign $\gamma_{\ell,p}$ number of big jobs from the beginning of list L_p to (M_k, u_ℓ), and remove these jobs from the list.

   Assignment of small jobs: let h_ℓ be the smallest number of small jobs from the beginning of L with a total processing time of at least $g_\ell(\varepsilon^2 u_q)$, and let k_ℓ be the maximum number of small jobs from the beginning of L that can be assigned to u_ℓ without violating the resource constraint (big jobs are taken into consideration). Assign $\min\{h_\ell, k_\ell\}$ jobs from the beginning of L to supply date u_ℓ on M_k, and remove them from L.

Proposition 1. *Each schedule S^{act} found by Algorithm A is feasible.*

Proof. Recall that a schedule is feasible if (i) on every machine the jobs do not overlap in time, and if (ii) $\sum_{(\ell\,:\,u_\ell \leq t)} \tilde{b}_\ell \geq \sum_{(j\,:\,S_j \leq t)} a_j$ for any $t \geq 0$.

In step 2 each examined tuple (partial assignment) must satisfy (2), thus Subroutine Sch guarantees (i) and (ii) for S^{part} (see Remark 1). Since we invoke the algorithm of the Appendix with at least $C_{\max}^{part}(k)$ amount of preassigned work on M_k ($k = 1, 2, \ldots, m$), S^{act} must satisfy (i).

The jobs scheduled at step 4 start after u_q, thus (ii) follows form Corollary 1. □

Proposition 2. *The running time of Algorithm A is polynomial in the size of the input.*

Proof. As described before, the number of tuples T can be bounded by $\binom{m+k_2}{k_2} = O((m+k_2)^{k_2}) = O((m + (1/\varepsilon)^{1/\varepsilon \cdot \log_{1+\varepsilon}(1/\varepsilon)})^{(1/\varepsilon)^{(1/\varepsilon \cdot \log_{1+\varepsilon}(1/\varepsilon))}})$, which is polynomial in m. Step 2 (Algorithm Assign) and step 3 require $O(n \log n)$ time, while step 4 also requires polynomial time ([12] and [11] or the Appendix). □

Let S^* be an optimal schedule, such that if $p_j = p_k$ and $a_j < a_k$, then $S_j^* \leq S_k^*$, $\forall j, k \in \mathcal{J}$. Such a schedule obviously exists. We construct an intermediate schedule $\tilde{S}$ from S^* to prove that our algorithm is a PTAS. This intermediate schedule has a similar structure to S^* and it is not far from a schedule computed by Algorithm A: (i) we use the big job arrangement of S^* to determine a tuple that we will use for $\tilde{S}$, (ii) the starting times of the big/huge jobs in $\tilde{S}$ will be determined by their starting times in S^* and (iii) the total processing time of the small jobs that start on M_k in $[u_\ell, u_{\ell+1})$ in $\tilde{S}$ is close to that in

S^*. The next paragraphs give a precise definition for $\tilde{S}$, along with an example.

To create $\tilde{S}$, first we perform steps 2 and 3 of Algorithm A with a tuple $T^* = (t_1^*, \dots, t_{k_2}^*)$ that we obtain from S^* in the following way: for each machine M_k we determine an assignment, which for each supply date ($\ell = 1, \dots, q$) and big job type ($p = 1, \dots, k_1$) describes the number of big jobs from $\mathcal{B}_p$ that start in $[u_\ell, u_{\ell+1})$ and $g_{\ell k}^*$, which is the smallest integer such that $(g_{\ell k}^* - 1) \cdot (\varepsilon^2 u_q)$ is at least the total processing time of small jobs starting in $[u_\ell, u_{\ell+1})$ on M_k in S^*. After that, we simply count the the number of the different assignments and determine T^*: there are t_i^* number of machines with assignment A_i ($i = 1, 2, \dots, k_2$). We can assume that machines $M_1, \dots, M_{t_1^*}$ are with assignment A_1, machines $M_{t_1^*+1}, \dots, M_{t_1^*+t_2^*}$ are with A_2, etc.

Note that we have to choose $g_{\ell k}^*$ in a special way for technical reasons (cf. Observation 2). Let $\tilde{S}^{part}$ denote the resulting partial schedule and $\tilde{C}_{\max}^{part}$ its makespan.

Example 2. *Let $m = 2$, $q = 3$ and $k_1 = 2$ and suppose that the schedule S^* shown in Figure 2 (above) is optimal. First we determine T^*: the assignment of M_1 is $(\Gamma_1^i = (2, 0, g_{11}^*), \Gamma_2^i = (1, 2, g_{21}^*))$, while the assignment of M_2 is $(\Gamma_1^{i'} = (0, 1, g_{12}^*), \Gamma_2^{i'} = (0, 0, g_{22}^*))$, where for example the first two coordinates of Γ_1^i shows that there are two jobs from $\mathcal{B}_1$ on M_1 that start in $[u_1, u_2)$ (noted by a '+') and there is no job from $\mathcal{B}_2$ that starts there. g_{11}^* describes roughly the total amount of processing time of the small jobs that start in $[u_1, u_2)$ on M_1: if this value is, say,* 80 *and $\varepsilon^2 u_q = 10$, then $g_{11}^* = 9$, since this is the smallest integer such that $(g_{11}^* - 1) \cdot (\varepsilon^2 u_q) \geq 80$.*

After we have created $\tilde{S}^{part}$, let $\tilde{J}_a$ denote the set of the unscheduled jobs. Schedule the remaining big and huge jobs at $\tilde{S}_j := S_j^* + 4\varepsilon u_q$ on the same machine as in S^*. See the third schedule ($\tilde{S}'$) in Figure 2 for an illustration.

Remark 2. *Note that we have $S_j^* \geq u_q$ for these jobs. For the huge jobs we have assumed this, while the for the big jobs this follows from the assumption about S^* (after Proposition 2): for each $p \in \{1, 2, \dots, k_1\}$ the jobs in $\tilde{J}_a \cap \mathcal{B}_p$ are the jobs in $\mathcal{B}_p$ with $S_j^* \geq u_q$, since these are the jobs from $\mathcal{B}_p$ with the biggest a_j value (cf. step 1 of Algorithm Assign).*

Finally schedule the remaining small jobs in arbitrary order after $\max\{u_q, C_{\max}^{part}\}$ at the earliest idle time on any machine (see the paragraph before Claim 1). Let $\tilde{C}_{\max}(k)$ denote the completion time of the last job scheduled on M_k in $\tilde{S}$ and $\tilde{C}_{\max} := C_{\max}(\tilde{S}) = \max_{k \in \mathcal{M}} \tilde{C}_{\max}(k)$.

Now we start the main part of the proof: we will prove that the makespan of $\tilde{S}$ is near to the makespan of S^{best} (Proposition 3) and to the makespan of S^* (Proposition 4). To prove these propositions we need to introduce some notations and prove a technical statement (Observation 2): let $\tilde{\mathcal{J}}_{\ell,k}$ denote the set of small jobs that are assigned

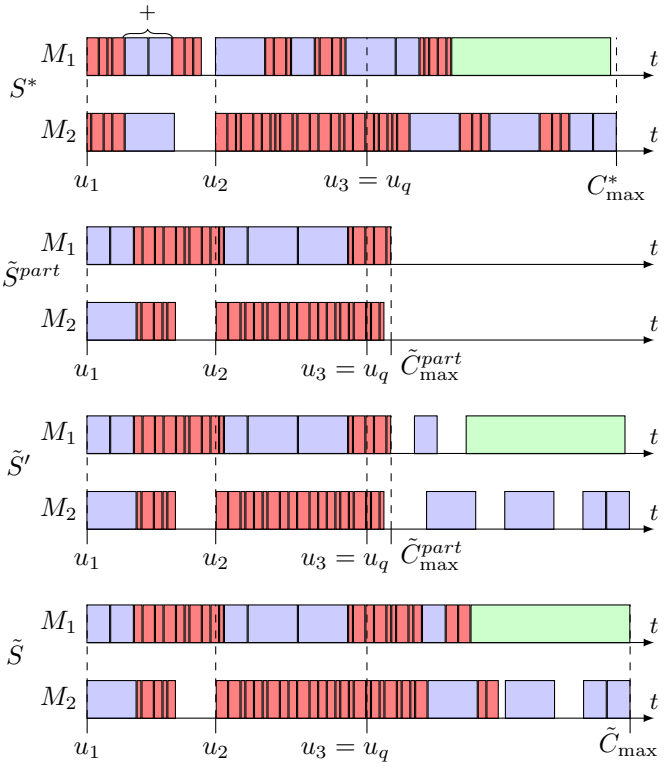

Figure 2: Creating $\tilde{S}$ from S^*. The small jobs are red, the big jobs are blue and we have a green huge job.

to u_ℓ and M_k in $\tilde{S}$ and $\mathcal{J}_{\ell,k}^*$ denote the set of small jobs with $u_\ell \leq S_j^* < u_{\ell+1}$ on machine k. $\tilde{\mathcal{J}}_\ell := \cup_k \tilde{\mathcal{J}}_{\ell,k}$ and $\mathcal{J}_\ell^* := \cup_k \mathcal{J}_{\ell,k}^*$.

Example 3. *Consider the schedules presented in Figure 2. From schedule S^*, we have, say, $p(\mathcal{J}_{1,1}^*) = 80$, $p(\mathcal{J}_{1,2}^*) =$* 45 *(hence $p(\mathcal{J}_1^*) =$* 125*) and $p(\mathcal{J}_{2,1}^*) = 65$, $p(\mathcal{J}_{2,2}^*) = 195$ ($p(\mathcal{J}_2^*) =$* 260*). The small jobs of $\mathcal{J}_{1,1}^*$ and $\mathcal{J}_{2,1}^*$ are in two-two contiguous parts. Suppose that $\varepsilon^2 u_q = 10$. From the definition of g^*, we can obtain $g_{11}^* = 9$, $g_{12}^* = 6$, $g_{21}^* = 8$ and $g_{22}^* = 21$.*

From $\tilde{S}^{part}$, we can determine $\tilde{\mathcal{J}}_{\ell,k}$ and $\tilde{\mathcal{J}}_\ell$: for example $\tilde{\mathcal{J}}_{1,1}$ is the set of the small jobs that start between the second and the third big job on M_1, while $\tilde{J}_{2,2}$ is the set of the small jobs that start after u_2 on M_2 in $\tilde{S}^{part}$.

By the rules of Algorithm A, we have the following:

Observation 2. *For each $\ell < q$ and $M_k \in \mathcal{M}$, $\sum_{j \in \tilde{\mathcal{J}}_{\ell,k}} p_j < \sum_{j \in \mathcal{J}_{\ell,k}^*} p_j + 3\varepsilon^2 u_q$ and $\sum_{j \in \cup_{\nu \leq \ell} \tilde{\mathcal{J}}_\nu} p_j \geq \sum_{j \in \cup_{\nu \leq \ell} \mathcal{J}_\nu^*} p_j - \varepsilon^2 u_q$.*

Proof. The first part follows from

$$\sum_{j \in \tilde{\mathcal{J}}_{\ell,k}} p_j < (g_{\ell k}^* + 1) \cdot (\varepsilon^2 u_q) < \sum_{j \in \mathcal{J}_{\ell,k}^*} p_j + 3\varepsilon^2 u_q,$$

where the first inequality follows from construction of $\tilde{S}$ (step 2 of Algorithm Assign), since we define h_ℓ so that

$\sum_{j\in\tilde{\mathcal{J}}_{\ell,k}} p_j$ cannot be greater than $g^*_{\ell k}\cdot(\varepsilon^2 u_q)+\varepsilon^2 u_q$, where the last $\varepsilon^2 u_q$ comes from the maximal processing time of a small job. The second part follows from the choice of g^*.

For the second part, note that for each $\nu \le q$ the big jobs assigned to a time point before u_ν in $\tilde{S}$ require at most the same amount of resource as the big jobs which start before u_ν in S^* (cf. steps 1 and 2 of Algorithm Assign). Thus, we have at least the same amount of resource for the small jobs until each u_ν in $\tilde{S}$ as in S^*. In the course of assigning the small jobs in $\tilde{S}$ there can be two reasons for switching to the next supply date (cf. step 2): (i) there is not enough resource to schedule the next small job from the list, or (ii) we reach the total required processing time. In the first case, we have $\sum_{j\in\cup_{\nu\le\ell}\mathcal{J}^*_\nu} p_j \le \sum_{j\in\cup_{\nu\le\ell}\tilde{\mathcal{J}}_\nu} p_j + \varepsilon^2 u_q$, since the small jobs are in non-increasing p_j/a_j order in L (step 1). Otherwise in case (ii), for each machine k, the algorithm assigns at least $g^*_{\nu k}\cdot(\varepsilon^2 u_q)$ amount of work from the small jobs to u_ν, thus $\sum_{j\in\tilde{\mathcal{J}}_\nu} p_j \ge \sum_{j\in\mathcal{J}^*_\nu} p_j + m\varepsilon^2 u_q$, and the observation follows. □

Let $C^*_q(k)$ denote the maximum of u_q and the completion time of the last job scheduled before u_q on M_k in S^*.

Corollary 2. *$\tilde{C}^{part}_{\max}(k) \le C^*_q(k) + 4\varepsilon u_q, \quad \forall k \in \mathcal{M}$.*

Proof. Since the big job layouts are the same in $\tilde{S}$ and in S^*, we have $\tilde{C}^{part}_{\max}(k) - C^*_q(k) \le \sum_{\ell=1}^{q-1}(\sum_{j\in\tilde{\mathcal{J}}_{\ell,k}} p_j - \sum_{j\in\mathcal{J}^*_{\ell,k}} p_j) < (q-1)\cdot(3\varepsilon^2 u_q) \le 4\varepsilon u_q$ for each $k \in \mathcal{M}$, where the second inequality follows from the first part of Observation 2. □

Proposition 3. *$\tilde{S}$ is feasible, and $C_{\max}(S^{best}) \le (1+\varepsilon)\tilde{C}_{\max}$.*

Proof. The feasibility of $\tilde{S}^{part}$ follows from Proposition 1, while Remark 2 and Corollary 2 show that the jobs in $\tilde{J}_a$ start after u_q, thus $\tilde{S}$ satisfies the resource constraints. Corollary 2 also guarantees that, the jobs in $\cup_{\ell<q}\tilde{\mathcal{J}}_{\ell,k}$ must end before a big or a huge job scheduled on M_k in the last stage of the construction of $\tilde{S}$ would start, since for all those big and huge jobs, $\tilde{S}_j = S^*_j + 4\varepsilon u_q$ by definition. Since the jobs cannot overlap in time after we insert a job into a partial schedule (Claim 1), $\tilde{S}$ is feasible.

In some iteration, Algorithm A will consider the tuple T^* that we used to define $\tilde{S}$. Hence, after step 3, $\tilde{S}^{part}$ and S^{part} coincide. Therefore, the Proposition follows from a result of [11], which is repeated in the Appendix for the sake of completeness. □

Proposition 4. *$\tilde{C}_{\max} \le C^*_{\max} + 5\varepsilon u_q$.*

Proof. Let j be such that $\tilde{C}_j = \tilde{C}_{\max}$ and let j be scheduled on M_k in $\tilde{S}$. If $j \notin \tilde{J}_a$, then the proposition follows from the first part of Observation 2 and from the fact that layouts of the big jobs (not in $\tilde{J}_a$) on M_k are the same in S^* and in $\tilde{S}$.

If $j \in \tilde{J}_a$ and j is big or huge, then originally we have $\tilde{S}_j = S^*_j + 4\varepsilon u_q$ and we may push j to the right by at most $\varepsilon^2 u_q$, thus $\tilde{C}_j \le C^*_{\max} + 5\varepsilon u_q$. If j is small, then the finishing time of each machine is in $[\tilde{S}_j - \varepsilon^2 u_q, \tilde{S}_j]$, otherwise j would be scheduled on another machine. For similar reasons, there is no idle time on any machine in $[\max\{\tilde{C}^{part}_{\max}(k)(k), u_q\}, \tilde{S}_j - \varepsilon^2 u_q]$ in $\tilde{S}$. Therefore,

$$\tilde{C}_{\max} \le \frac{\sum_{k=1}^m \tilde{C}_{\max}(k)}{m} + \varepsilon^2 u_q = \\ = \frac{\sum_{k=1}^m \max\{\tilde{C}^{part}_{\max}(k)(k), u_q\} + p(\tilde{J}_a)}{m} + \varepsilon^2 u_q, \quad (5)$$

where $p(\tilde{J}_a) := \sum_{j\in\tilde{J}_a} p_j$. To sum up, we have

$$C^*_{\max} \ge \frac{\sum_{k=1}^m C^*_q(k) + \sum_{j:S^*_j\ge u_q} p_j}{m} \ge \\ \ge \frac{\sum_{k=1}^m \max\{\tilde{C}^{part}_{\max}(k)(k), u_q\} - 4m\varepsilon u_q + (p(\tilde{J}_a) - \varepsilon^2 u_q)}{m} \ge \\ \ge \tilde{C}_{\max} - 5\varepsilon u_q,$$

where the first inequality is trivial, the second follows from Corollary 2 and from the second part of Observation 2 (if $\ell = q-1$) and the third from (5). The proposition follows. □

Proof of Theorem 1. We have seen that Algorithm A is polynomial (Proposition 2) and creates a feasible schedule (Proposition 1) with a makespan $C_{\max}(S^{best}) \le (1+\varepsilon)\tilde{C}_{\max} \le (1+\varepsilon)(C^*_{\max} + 5\varepsilon u_q) \le (1+7\varepsilon)C^*_{\max}$ (Propositions 3 and 4), thus it is a PTAS. □

Acknowledgments

The author would like to thank Tamás Kis for comments that greatly improved the manuscript. This work has been supported by the OTKA grant K112881, and by the GINOP-2.3.2-15-2016-00002 grant of the Ministry of National Economy of Hungary.

References

[1] F. Belkaid, F. Maliki, F. Boudahri, and Z. Sari. A branch and bound algorithm to minimize makespan on identical parallel machines with consumable resources. In *Advances in Mechanical and Electronic Engineering*, pages 217–221. Springer, 2012.

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

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

[4] S. Carrera, W. Ramdane-Cherif, and M.-C. Portmann. Scheduling supply chain node with fixed component arrivals and two partially flexible deliveries. In *5th International Conference on Management and Control of Production and Logistics-MCPL 2010*, page 6. IFAC Publisher, 2010.

[5] 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.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, LA: Freeman, 1979.
[7] A. Grigoriev, M. Holthuijsen, and J. van de Klundert. Basic scheduling problems with raw material constraints. *Naval Research of Logistics*, 52:527–553, 2005.
[8] P. Györgyi and T. Kis. Approximation schemes for single machine scheduling with non-renewable resource constraints. *Journal of Scheduling*, 17:135–144, 2014.
[9] P. Györgyi and T. Kis. Reductions between scheduling problems with non-renewable resources and knapsack problems. *Theoretical Computer Science*, 565:63–76, 2015a.
[10] P. Györgyi and T. Kis. Approximability of scheduling problems with resource consuming jobs. *Annals of Operations Research*, 235(1):319–336, 2015b.
[11] P. Györgyi and T. Kis. Approximation schemes for parallel machine scheduling with non-renewable resources. *European Journal of Operational Research*, 258(1):113 – 123, 2017.
[12] L. A. 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] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.
[14] E. Morsy and E. Pesch. Approximation algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 18(6):645–653, 2015.
[15] R. Slowinski. Preemptive scheduling of independent jobs on parallel machines subject to financial constraints. *European Journal of Operational Research*, 15:366–373, 1984.
[16] A. Toker, S. Kondakci, and N. Erkip. Scheduling under a nonrenewable resource constraint. *Journal of the Operational Research Society*, 42:811–814, 1991.
[17] J. Xie. Polynomial algorithms for single machine scheduling problems with financial constraints. *Operations Research Letters*, 21(1):39–42, 1997.

Appendix, A PTAS for $P|preassign, r_j|L_{\max}$

In this section we sketch how to extend the PTAS of Hall and Shmoys [12] for parallel machine scheduling with release dates, due-dates and the maximum lateness objective ($P|r_j|L_{\max}$) with pre-assigned works on the machines. The jobs scheduled on a machine must succeed any pre-assigned work.

Hall and Shmoys propose an $(1+\varepsilon)$-optimal outline scheme in which job sizes, release dates, and due-dates are rounded such that the schedules can be labeled with concise outlines, and there is an algorithm which given any outline ω for an instance I of the scheduling problem, delivers a feasible solution to I of value at most $(1+\varepsilon)$ times the value of any feasible solutions to I labeled with ω.

All we have to do to take pre-assigned work into account is that we extend the outline scheme of Hall and Shmoys with machine ready times, which are time points when the machines finish the pre-assigned work. Suppose the largest of these time points is $w_{\max}$. We divide $w_{\max}$ by $\varepsilon/2$ and round each of the pre-assigned work sizes of the machines down to the nearest multiple of $2w_{\max}/\varepsilon$. Thus the number of distinct pre-assigned work sizes is $2/\varepsilon$, a constant independent of the number of jobs and machines. Then, we amend the machine configurations (from which outlines are built) with the possible rounded pre-assigned work sizes. Finally, the algorithm which determines a feasible solution from an outline must be modified such that it disregards all the outlines in which any job is scheduled on a machine before the corresponding rounded pre-assigned work size in the outline, and if the rounded pre-assigned work sizes of the outline do not match the real pre-assigned works of the machines.