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

Péter Györgyi\textsuperscript{a,b}

\textsuperscript{a}Department of Operations Research, Loránd Eötvös University, H1117 Budapest, Pázmány Péter sétány 1/C, Hungary
\textsuperscript{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, \( M = \{M_1, \ldots, M_m\} \), a finite set of \( n \) jobs \( J = \{J_1, \ldots, J_n\} \), 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 \( b_\ell \in \mathbb{Z}_+ \) represents the quantity supplied at \( u_\ell \), \( \ell = 1, 2, \ldots, q \). A schedule \( \sigma \) 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., \( \sum_{\ell : u_\ell \leq t} b_\ell \geq \sum_{j : S_j \leq t} a_j \). 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 \( b_\ell \) are \( r \)-dimensional vectors. We denote our problem by \( P|\text{rm} = 1|C_{\text{max}} \), where \( \text{rm} = 1 \) 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:

\textbf{Theorem 1.} There is a PTAS for \( P|\text{rm} = 1|C_{\text{max}} \).

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, when 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 \( P|\text{rm} = 1|C_{\text{max}} \) was the only problem with unknown approximability status in the class \( P|\text{rm} = 1|C_{\text{max}} \) ([11]).

Our PTAS reuses ideas from known PTAS-es designed for \( P|C_{\text{max}} \) (e.g. [13],[12]). Actually, we invoke a variant of the latter. However, there are no resource constraints in...
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 $\Pi$ 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 $\varepsilon$-approximation algorithm for an optimization problem $\Pi$ delivers in polynomial time for each instance of $\Pi$ 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 $\Pi$, a family of approximation algorithms $\{A_{\ell}\}_{\ell>0}$, where each $A_{\ell}$ is an $\varepsilon$-approximation algorithm for $\Pi$ is called a Polynomial Time Approximation Scheme (PTAS) for $\Pi$.

2. Previous work

Makespan minimization on parallel machine 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, 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 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-s 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 $C^*_{\max} > u_q$ follows from this assumption:

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

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

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

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

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

s.t.

$$\sum_{j \in \mathcal{J}} \sum_{\ell} a_{j} x_{j\ell k} \leq b_{\ell}, \quad u_\ell \in \mathcal{T}$$

$$\sum_{j \in \mathcal{J}} x_{j\ell k} = 1, \quad j \in \mathcal{J}$$

$$x_{j\ell k} \in \{0,1\}, \quad j \in \mathcal{J}, \quad u_\ell \in \mathcal{T}, \quad k \in \mathcal{M}.$$
scheduled in \( \tilde{S} \), and we schedule \( j_1 \) on \( M_k \) with starting time \( t_1 \in I \). This transforms \( \tilde{S} \) as follows. For each job \( j \) scheduled on \( M_k \) in \( \tilde{S} \) with \( \tilde{S}_j = t_1 \), let \( P_k [t_1, \tilde{S}_j] \) denote the total processing time of those jobs scheduled on \( M_k \) in \( \tilde{S} \) between \( t_1 \) and \( \tilde{S}_j \). We update the start-time of \( j \) to max\( \{ \tilde{S}_j + p_j + P_k [t_1, \tilde{S}_j] \} \). The start time of all other jobs do not change.

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

In this section we prove Theorem 1. Let \( \varepsilon > 0 \) be fixed. It is enough to deal with the case where \( q = \lceil 1/\varepsilon \rceil + 1 \) (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 \( J \) into three subsets. A job \( j \) is small \( (j \in S) \), if \( p_j \leq \varepsilon^2 u_q \), it is big \( (j \in B) \), if \( \varepsilon^2 u_q < p_j < (1/\varepsilon) u_q \) and it is huge \( (j \in H) \), 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 \( \varepsilon \) fraction of the makespan.

Note that there are \( k_1 := \lceil 1 + 3 \log_{1+\varepsilon} (1/\varepsilon) \rceil \) possible values for the processing time of a big job, let \( B_1, \ldots , B_{k_1} \) denote the sets of the big jobs with the same processing times.

Consider an arbitrary pair \( (M_k, u_q) \), where \( k \in \{ 1, \ldots , m \} \) and \( \ell \in \{ 1, \ldots , q-1 \} \). We guess the number of the big jobs from each type and the total processing time of the small jobs in the form of \( g_{k, \ell} \cdot (\varepsilon^2 u_q) \) that start in \([u_{\ell}, u_{\ell+1})\) on \( M_k \). A guess is a tuple \( (t_{k, \ell,1}, t_{k, \ell,2}, \ldots , t_{k, \ell, k_1}, g_{k, \ell}) \), where each coordinate is from the set \( \{ 0, 1, \ldots , \lceil 1/\varepsilon \rceil + 1 \} \). The number of the different guesses for a given machine \( k \) is at most \( k_2 := \lceil (1/\varepsilon + 2) (q-1)(k_1+1) \rceil \).

Another tuple \( T^{*} := (t^{*}_{1}, t^{*}_{2}, \ldots , t^{*}_{m}) \) 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 \( (m+k_2) \), thus it is polynomial. We examine all assignments and either create a schedule according to the assignment or declare that the assignment is unfeasible.

The remaining big jobs are assigned to \( u_q \), but we assign them to machines later, while we will use a greedy algorithm to define the assignment of the small jobs according to the guess. The algorithm is as follows:

**Algorithm A**

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

1. For each tuple \( T' = (t'_1, \ldots , t'_m) \), 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 \( S^{\text{part}} \) from \( \hat{x} \) with Subroutine Sch ([11], Appendix B). Let \( C_{\text{max}} (k) \) be the time when \( M_k \) finishes \( S^{\text{part}} \).
4. Invoke the algorithm of Appendix C with \( \max \{ C_{\text{max}} (k), u_q \} \) amount of preassigned work on \( M_k \) \((k = 1, 2, \ldots , m)\) to schedule the remaining jobs. Let \( S^{\text{act}} \) be the resulting schedule.
5. If \( C_{\text{max}} (S^{\text{act}}) < C_{\text{max}} (S^{\text{best}}) \), then let \( S^{\text{best}} := S^{\text{act}} \).
6. After examining each feasible assignment before \( u_q \), output \( S^{\text{best}} \).

The next algorithm assigns jobs to pairs \( (M_k, u_{\ell}) \), where \( \ell < q \). Machines \( M_1, \ldots , M_t \) get assignment \( A_1, M_{t+1}, \ldots , M_{t+q-1} \) get \( A_2 \), etc.

**Algorithm Assign**

Input: a tuple \( T' \). Output: a (partial) assignment \( \hat{x} \).

1. For each \( p = 1, \ldots , k_1 \), order big jobs from \( B_p \) 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. For each \( p = 1, \ldots , k_1 \), assign the required number of big jobs from the beginning of the list \( L_p \) to the machinesupply date pairs in the following order: \( (M_1, u_1), (M_2, u_1), \ldots , (M_m, u_1) \), \( (M_1, u_2), \ldots , (M_m, u_2) \), \( (M_1, u_3), \ldots , (M_m, u_{q-1}) \) (after a job is assigned to a pair \( (M_k, u_{\ell}) \) then remove it from its list).
3. Let \( h_{1,1} \) be the smallest number of small jobs from the beginning of \( L \) with a total processing time of at least \( g_{1,1} (\varepsilon^2 u_q) \), and let \( k_{1,1} \) be the maximum number of small jobs from the beginning of \( L \) that can be assigned to \( u_1 \) without violating the resource constraint (big jobs are taken into consideration). Assign \( \min \{ h_{1,1}, k_{1,1} \} \) jobs from the beginning of \( L \) to supply date \( u_1 \) on \( M_1 \), and remove them from \( L \). Then proceed with the next machine until all machines are processed. Then proceed with \( u_2 \) etc, until all the supply dates from \( u_1 \) through \( u_{q-1} \) are processed.

The final schedule \( S^{\text{best}} \) is obviously feasible and the running time of the algorithm is polynomial in the size if the input, since the number of tuples \( T' \) can be bounded by \( \left( m + k_2 \right)^{m+k_2} \), step 2 (Algorithm Assign) and step 3 require \( O\left( n \log n \right) \) time, while step 4 also requires polynomial time ([12] and [11] or Appendix C).

We construct a schedule \( \tilde{S} \) from an optimal schedule \( S^* \) to prove that our algorithm is a PTAS. To create \( \tilde{S} \), first we perform steps 2 and 3 of Algorithm A with a tuple \( T^{*} \) that corresponds to \( S^* \) in the following way: there are \( t_{\mu} \) number of machines with assignment \( A_\mu \) \((\mu = 1, 2, \ldots , k_2)\) in \( S^* \), where an assignment describes the number of different big jobs assigned to the supply dates and \( g_{k, \ell} (\ell < q) \), which is the smallest integer such that \( (g_{k, \ell} - 1) \cdot (\varepsilon^2 u_q) \) is at least the total processing time of small jobs starting in \( [u_{\ell}, u_{\ell+1}) \) on \( M_k \) in \( S^* \). We can assume that each \( M_k \) has the same assignment in \( S^* \) and in \( \tilde{S} \). After that, let \( J_0 \) denote the set of the unscheduled jobs. Schedule the remaining big and huge jobs at \( \tilde{S}_j := S^* + 4 \varepsilon u_q \) on the same machine as in \( S^* \) and finally schedule the remaining small jobs in arbitrary order after \( \max \{ u_q, C_{\text{max}} \} \) at the earliest idle time on any machine. Let \( C_{\text{max}} := C_{\text{max}} (\tilde{S}) \).

Let \( J_{t, k} \) denote the set of small jobs that are assigned to \( u_{\ell} \) and \( M_k \) in \( \tilde{S} \) and \( J_{t, k} \) denote the set of small jobs
with \( u_t \leq S_j^* < u_{t+1} \) on machine \( k \). \( \tilde{J}_k := \cup_t \tilde{J}_k \) and \( \bar{J}_k := \cup_t \bar{J}_k \). By the rules of Algorithm A, we have the followings:

Observation 2. For each \( \ell < q \) and \( M_k \in \mathcal{M} \), \( \sum_{j \in \bar{J}_k} p_j + 3\epsilon^2 u_q \) and \( \sum_{j \in \cup_t \tilde{J}_k} p_j \geq \sum_{j \in \cup_t \tilde{J}_k} p_j - \epsilon^2 u_q \).

Proof. The first part follows from the choice of \( g^* \) and the construction of \( S \). For the second part, note that for each \( \nu \leq q \) the big jobs assigned to a time point before \( u_\nu \) in \( \tilde{S} \) require at most the same amount of resource as the big jobs start before \( u_\nu \) in \( S^* \). Thus, we have at least the same amount of resource for the small jobs until each \( u_\nu \) 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: (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 \( \sum_{j \in \cup_t \tilde{J}_k} p_j \geq \sum_{j \in \cup_t \tilde{J}_k} p_j + \epsilon^2 u_q \), since the small jobs are in non-increasing \( p_j/u_j \) order in \( L \). Otherwise in case (ii), for each machine \( k \), the algorithm assigns at least \( g_{j_0} \cdot (\epsilon^2 u_q) \) amount of work from the small jobs to \( u_\nu \), thus \( \sum_{j \in \tilde{J}_k} p_j \geq \sum_{j \in \tilde{J}_k} p_j + m\epsilon^2 u_q \), and the observation follows.

Let \( C^*_v(k) \) denote the maximum of \( u_q \) and the completion time of the last job scheduled before \( u_q \) on \( M_k \) in \( S^* \). The next statement is an easy corollary of the first part of Observation 2.

Corollary 1. \( C^\text{part}_{\text{max}}(k) \leq C^*_v(k) + 4\epsilon u_q, \quad \forall k \in \mathcal{M}. \)

Proposition 1. \( \tilde{S} \) is feasible, and \( C^\text{max}(S^\text{best}) \leq (1 + \epsilon)\tilde{C}^\text{max}. \)

Proof. \( \tilde{S} \) cannot violate the resource constraints by the rules of Algorithm A, and due to Corollary 1, the jobs in \( \cup_t \tilde{J}_k \) 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, \( \tilde{S}_j = S^*_j + 4\epsilon u_q \) by definition. In some iteration, Algorithm A will consider the tuple that we used to define \( \tilde{S} \). Hence, after step 3, \( \tilde{S} \) and \( S^\text{part} \) coincide. Therefore, the Proposition follows from a result of [11], which is repeated in Appendix C for the sake of completeness.

Proposition 2. \( \tilde{C}^\text{max} \leq C^\text{max} + 5\epsilon u_q. \)

Proof. Let \( j \) be such that \( \tilde{C}^*_j = \tilde{C}^\text{max} \). If \( j \notin \tilde{J}_a \), then the proposition follows from the first part of Observation 2.

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

Note that \( P \leq \sum_{j \notin \tilde{J}_a} p_j + \epsilon^2 u_q \) (second part of Observation 2). If an arbitrary job \( j' \) is scheduled on \( M_k \) in \( S^* \) with \( S^*_j > u_q \), then \( S^*_j \geq \max\{C^\text{part}_{\text{max}}(k), u_q\} - 4\epsilon u_q \) (Corollary 1), thus \( C^\text{max} \geq \sum_{k}(\max\{C^\text{part}_{\text{max}}(k), u_q\} - 4\epsilon u_q \geq u_q + p(\tilde{J}_a) - \epsilon^2 u_q) / m \geq \tilde{C}^\text{max} - 5\epsilon u_q \) and the proposition follows.

Proof of Theorem 1. We have seen that Algorithm A is polynomial and creates a feasible schedule with a makespan \( (S^\text{best}) \leq (1 + \epsilon)\tilde{C}^\text{max} \leq (1 + \epsilon)(C^\text{max} + 5\epsilon u_q) \leq (1 + 7\epsilon)\tilde{C}^\text{max} \) (Propositions 1 and 2), 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.

References


The amount of resource(s) supplied at instance same.

For the third follows from one is the crux of the derivation and will be shown below, \( A_\varepsilon \) (a constant for any fixed \( \varepsilon \)). Notice that for each instance the following condition:

\[
\text{A}_\varepsilon \text{ applied to } I' \implies \text{the first inequality. The second one is the crux of the derivation and will be shown below, the third follows from } u_q \leq C_{\text{max}}(I). \]

By Observation 1, the above derivation implies that we get a PTAS for \( P[\text{rm}][C_{\text{max}}] \).

Suppose that there are \( q \) supplies in instance \( I \) of \( P[\text{rm}][C_{\text{max}}] \) with quantities \( b_1, b_2, \ldots, b_q \). We construct instance \( I' \) of the restricted problem: the \( q' := \lceil 1/\varepsilon \rceil + 1 \) (a constant for any fixed \( \varepsilon \)) supply dates are \( u'_1 = 0 \), \( u'_\ell = (\ell - 1)\varepsilon u_q \) for \( \ell = 2, \ldots, q' - 1 \), and \( u'_{q'} = u_q \). The amount of resource(s) supplied at \( u'_1 \) is \( b'_1 := b_1 \), and for \( u'_\ell \) with \( \ell \geq 2 \) it is \( b'_\ell = \sum_{b_k, u_k \leq \varepsilon u_q} b_k - \sum_{b_k, u_k > \varepsilon u_q} b_k \) (see Figure 1). Notice that for each \( u'_\ell \) there is an \( u'_{\ell'} \) with \( u'_{\ell'} \leq u'_\ell \). Besides, the two instances are the same.

\[
\begin{array}{cccccccc}
& b_1 & b_2 & b_3 & b_4 & b_5 & b_\ell-1 & b_\ell \\
u_1 = 0 & u_2 & u_3 & u_4 & u_5 & \ldots & \ & u_{\ell-1} & u_\ell \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\ell' & b'_1 & b'_2 & b'_3 & b'_4 & b'_5 & b'_{\ell-1} & b'_{\ell} \\
u'_1 = 0 & u'_2 & u'_3 & u'_4 & u'_5 & \ldots & \ & u'_{\ell-1} & u'_{\ell} \\
\end{array}
\]

Figure 1: Supplies in case of an instance with an arbitrary number of supplies (above) and the corresponding instance with constant number of supplies (below).

Let \( S'_I \) be an optimal schedule for \( I \). If we increase the starting time of each job by \( \varepsilon u_q \), then the resulting schedule is a feasible solution of instance \( I' \), since the supplies are delayed by less than \( \varepsilon u_q \). Hence, by using the properties of \( C_{\text{max}}, C_{\text{max}}(I') \leq C_{\text{max}}(I) + \varepsilon u_q \) follows.

Appendix B, Subroutine Sch

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

Output: partial schedule \( S_{\text{part}} \) of the jobs in \( J' \).

1. \( S_{\text{part}} \) is initially empty, then we schedule the jobs on each machine in increasing \( u_t \) 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\{\( u_t \), \( C_{\text{last}}(k) \)\}, where \( C_{\text{last}}(k) \) is the completion time of the last job scheduled on machine \( M_k \), or 0 if no job has been scheduled yet on \( M_k \).

Appendix C, A PTAS for \( P[\text{preassign}, r_j|L_{\text{max}}] \)

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 \( P[\text{r}][L_{\text{max}}] \) 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 \( \omega \) 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 \( \omega \).

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 \( w_{\text{max}} \). We divide \( w_{\text{max}} / \varepsilon \) and round each of the pre-assigned work sizes of the machines down to the nearest multiple of \( 2w_{\text{max}} / \varepsilon \). Thus the number of distinct pre-assigned work sizes is \( \varepsilon / 2 \), 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.