Scheduling Parallel Machines with Inclusive Processing Set Restrictions and Job Release Times

Chung-Lun Li
Department of Logistics and Maritime Studies
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
Email: *lgtclli@polyu.edu.hk*

Xiuli Wang
School of Economics and Management
Nanjing University of Science and Technology
Nanjing, China
Email: *wangdu0816@163.com*

July 2008
Revised December 2008

Abstract

We consider the problem of scheduling a set of jobs with different release times on parallel machines so as to minimize the makespan of the schedule. The machines have the same processing speed, but each job is compatible with only a subset of those machines. The machines can be linearly ordered such that a higher-indexed machine can process all those jobs that a lower-indexed machine can process. We present an efficient algorithm for this problem with a worst-case performance ratio of 2. We also develop a polynomial time approximation scheme (PTAS) for the problem, as well as a fully polynomial time approximation scheme (FPTAS) for the case in which the number of machines is fixed.

Keywords: Scheduling; parallel machines; release times; worst-case analysis; polynomial time approximation scheme

1 Introduction

In many applications of parallel machine scheduling, machines have the same speed, but they differ from each other in their functionality. As a result, every job has a restricted set of machines to which it may be assigned, called its *processing set*, while the processing time of a job is independent of all the machines assigned to it; see, for example, [6, 20, 23]. One particular type of parallel machine scheduling problems with processing set restrictions, namely problems with *inclusive processing sets*, have received increasing attention recently. In this type of scheduling problems, a job's processing set is either a subset or superset of another job's processing set.

Scheduling problems with inclusive processing set restrictions have many applications. A classical application is in scheduling computer programs to multiple processors with memory constraints, where a job can only be assigned to a processor with memory capacity no less than the job's memory requirement [15, 16]. Inclusive processing sets also arise in the service industry when service providers differentiate their customers by categorizing them as platinum, gold, silver, and regular members. One method of providing differentiated service to these customers is to label customers (i.e., jobs) and servers (i.e., machines) with grade of service (GoS) levels, and allow a customer to be served by a server only when the GoS level of the customer is no less than the GoS level of the server [10]. Ou *et al.* [23] have described an application in cargo loading, where multiple loading/unloading cranes are working in parallel to load/unload cargoes of a vessel. The cranes have identical operating speed but different weight capacity limits. Each piece of cargo (i.e., job) can be handled by any crane (i.e., machine) with a weight capacity limit no less than the weight of the cargo. The objective is to finish loading/unloading the vessel at a minimal time duration.

A number of studies of scheduling problems with inclusive processing set restrictions have appeared in the literature. Most of those offline scheduling models with inclusive processing sets assume that every job is available at time 0. However, it is quite common in practice that jobs have different release dates/times. For example, in the cargo loading application mentioned above, it is not uncommon to have some "late come cargoes" still on the way to the cargo loading area (and therefore they are not yet available for loading) when the loading operations of a vessel have already started. Therefore, in this paper we analyze a scheduling model with job release times and inclusive processing set restrictions, and we focus on the development of polynomial-time approximation algorithms for our model.

Our problem can be described formally as follows: Given a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m parallel machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. Associated with each job J_j are a processing time $p_j > 0$, a release time $r_j \geq 0$, and a machine index $a_j \in \{1, 2, \ldots, m\}$. Job J_j becomes available for processing at its release time, and it can be processed by machine M_i if $i \geq a_j$. In other words, the machines are linearly ordered in such a way that M_i can process all those jobs that M_{i-1} can process ($i = 2, 3, \ldots, m$). We denote $S_i = \{J_j \mid a_j = i\}$ for $i = 1, 2, \ldots, m$. Then, $\mathcal{J} = S_1 \cup S_2 \cup \cdots \cup S_m$, and the jobs in S_i can be processed by any of $M_i, M_{i+1}, \ldots, M_m$. Job preemption is not permitted. The objective is to determine a feasible schedule σ such that the makespan, denoted $C_{\max}(\sigma)$, is minimized. We assume that all processing times and release times of jobs are integers. We denote our problem as $P \,|\, r_j,\, \mathit{incl.\,proc.\,sets} \,|\, C_{\max}$. When all job release times are zero, we denote the problem as $P \,|\, \mathit{incl.\,proc.\,sets} \,|\, C_{\max}$. When the number of machines, m, is fixed, we denote $P \,|\, r_j,\, \mathit{incl.\,proc.\,sets} \,|\, C_{\max}$ and $P \,|\, \mathit{incl.\,proc.\,sets} \,|\, C_{\max}$ as $Pm \,|\, r_j,\, \mathit{incl.\,proc.\,sets} \,|\, C_{\max}$ and $Pm \,|\, \mathit{incl.\,proc.\,sets} \,|\, C_{\max}$, respectively.

Note that if $m > n$, then clearly, there exists an optimal solution to $P \,|\, r_j,\, \mathit{incl.\,proc.\,sets} \,|\, C_{\max}$ in which no job is scheduled on $M_1, M_2, \ldots, M_{m-n}$ and therefore the $m - n$ least flexible machines can be completely ignored. Hence, throughout the paper, we assume that $m \leq n$.

Scheduling with job release times have been studied by many researchers (see, for example, [2, 17]). However, research on parallel machine problems with job release times is limited. Some of these works focus on problems with a min-sum objective (see, for example, [3, 25]). For problems with a min-max objective, Hall and Shmoys [7] have developed a polynomial time approximation scheme (PTAS) for the strongly NP-hard problem $P \,|\, r_j \,|\, L_{\max}$, where $L_{\max} = \max\{s_j + p_j + q_j\}$, s_j is the processing start time of J_j, and q_j is a given delivery time of J_j. Mastrolilli [21] has developed a more efficient PTAS for the same problem. Note that $P \,|\, r_j \,|\, C_{\max}$ is a special case of $P \,|\, r_j \,|\, L_{\max}$, and therefore, both PTASs developed by [7, 21] are applicable to $P \,|\, r_j \,|\, C_{\max}$.

It is well known that the classical parallel machine minimum makespan scheduling problem,

$P \mid\mid C_{\max}$, is NP-hard in the strong sense when the number of machines is not fixed [17]. Thus, problems $P \mid incl.\, proc.\, sets \mid C_{\max}$ and $P \mid r_j,\, incl.\, proc.\, sets \mid C_{\max}$ are also strongly NP-hard. The strong NP-hardness of these problems not only indicates the difficulty of developing polynomial-time optimal algorithms, but it also implies that it is impossible to develop fully polynomial time approximation schemes (FPTASs) for these problems unless $P = NP$. A few researchers have developed polynomial-time approximation algorithms for problem $P \mid incl.\, proc.\, sets \mid C_{\max}$. In fact, $P \mid incl.\, proc.\, sets \mid C_{\max}$ is a special case of the unrelated parallel machine scheduling problem $R \mid\mid C_{\max}$, and a polynomial-time 2-approximation algorithm (i.e., an algorithm which generates solutions with relative error guaranteed no more than 100%) have been developed by Lenstra *et al.* [19]. Shchepin and Vakhania [26] have further developed a polynomial-time algorithm for $R \mid\mid C_{\max}$ with an improved worst-case performance ratio of $2 - \frac{1}{m}$. There are several polynomial-time algorithms for problem $P \mid incl.\, proc.\, sets \mid C_{\max}$ with worst-case performance ratio better than $2 - \frac{1}{m}$. These include a $(2 - \frac{1}{m-1})$-approximation algorithm developed by Kafura and Shen [15], a $(2 - \frac{1}{m-1})$-approximation algorithm developed by Hwang *et al.* [10], and a $\frac{3}{2}$-approximation algorithm developed by Glass and Kellerer [5]. Ou *et al.*'s [23] have developed a $\frac{4}{3}$-approximation algorithm with a polynomial running time of $O\big((n+m)(\log nm)\log p_{\text{sum}}\big)$, where $p_{\text{sum}} = \sum_{j=1}^{n} p_j$, and a $(\frac{4}{3}+\epsilon)$-approximation algorithm with a strongly polynomial running time of $O\big((n+m)(\log nm)\log \frac{1}{\epsilon}\big)$, where ϵ is a positive constant which may be set arbitrarily close to zero. Under the assumption of $m \leq n$, their $\frac{4}{3}$-approximation algorithm and $(\frac{4}{3}+\epsilon)$-approximation algorithm have running time of $O(n \log n \log p_{\text{sum}})$ and $O(n \log n \log \frac{1}{\epsilon})$, respectively. They have also presented a PTAS for the problem. To the best of our knowledge, no research has been reported on problem $P \mid r_j,\, incl.\, proc.\, sets \mid C_{\max}$.

When the number of machines is fixed, problem $Pm \mid incl.\, proc.\, sets \mid C_{\max}$ becomes NP-hard in the ordinary sense. Horowitz and Sahni [8], Jansen and Porkolab [11], and Fishkin *et al.* [4] have developed FPTASs for problem $Rm \mid\mid C_{\max}$ (i.e., problem $R \mid\mid C_{\max}$ when the number of machines is fixed). Thus, their FPTASs can be applied to problem $Pm \mid incl.\, proc.\, sets \mid C_{\max}$. Ji and Cheng [12] have also proposed a different FPTAS for the same problem. Mastrolilli [22] has developed an FPTAS for the unrelated parallel machine scheduling problem when the number of machines is fixed, with the objective of minimizing the maximum flow time of jobs, i.e., $\max_j\{C_j - r_j\}$. However, there is no known FPTAS for problem $Pm \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$.

A few researchers have developed online algorithms for various variants of problem $P \,|\, incl.\, proc.\, sets \,|\, C_{\max}$; see [1, 13, 14, 24]. Some researchers have also studied preemptive scheduling models with inclusive processing set restrictions; see [9].

The rest of this paper is organized as follows: In Section 2 we present some important properties of problem $P \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$ and then develop an efficient 2-approximation algorithm for the problem. In Section 3 we present a PTAS for the problem. In Section 4 we present an FPTAS for the case where the number of machines is fixed. We draw some concluding remarks in Section 5.

2 Model Properties and Efficient Approximation Algorithms

We first present two lemmas, which cover some important properties of problem $P \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$:

Lemma 1 *There exists an optimal solution to $P \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$ in which the jobs processed by M_i are sequenced in nondecreasing order of release times for $i = 1, 2, \ldots, m$.*

Proof: The proof follows a straightforward adjacent job interchange argument, and the details are omitted. ▮

Lemma 2 *Consider a machine M_i and a job subset $\{J_{j_1}, J_{j_2}, \ldots, J_{j_h}\}$, where $a_{j_u} \le i$ ($u = 1, 2, \ldots, h$) and $r_{j_1} \le r_{j_2} \le \cdots \le r_{j_h}$. Let D be a positive integer. Suppose jobs $J_{j_2}, J_{j_3}, \ldots, J_{j_h}$ can all be scheduled on machine M_i with each job completion time no greater than D. Then, jobs $J_{j_1}, J_{j_2}, \ldots, J_{j_h}$ can all be scheduled on machine M_i with each job completion time no greater than D if and only if $r_{j_1} + p_{j_1} \le D - (p_{j_2} + p_{j_3} + \cdots + p_{j_h})$.*

Proof: Suppose that $r_{j_1} + p_{j_1} \le D - (p_{j_2} + p_{j_3} + \cdots + p_{j_h})$. Since $J_{j_2}, J_{j_3}, \ldots, J_{j_h}$ can all be scheduled on machine M_i with each job completion time no greater than D, by Lemma 1, we can schedule these jobs on M_i in nondecreasing order of release times with the completion time of the last job no greater than D. Hence, if we schedule J_{j_u} to start at time $D-(p_{j_u}+p_{j_{u+1}}+\cdots+p_{j_h})$ for $u=1,2,\ldots,h$, then jobs $J_{j_1},J_{j_2},\ldots,J_{j_h}$ can all be processed by machine M_i with no time clash and no violation of release time constraints. Conversely, suppose that $r_{j_1}+p_{j_1}>D-(p_{j_2}+p_{j_3}+\cdots+p_{j_h})$. Then, $r_{j_1}\geq D-(p_{j_1}+p_{j_2}+\cdots+p_{j_h})+1$, which implies that $J_{j_1},J_{j_2},\ldots,J_{j_h}$ must all be processed within the time interval $[D-(p_{j_1}+p_{j_2}+\cdots+p_{j_h})+1,\ D]$. Hence, these jobs cannot be all assigned to the same machine. ∎

A straightforward approach to obtaining an approximation solution for problem $P\,|\,r_j,\ \mathit{incl.proc.sets}\,|\,C_{\max}$ with a constant bound on the relative error is to apply the method developed by Ou *et al.* [23]. Such an approach is described as follows:

Algorithm $A1$:

Step 1. Ignore all job release times and assign the jobs to machines using Ou *et al.*'s $\frac{4}{3}$-approximation algorithm. Within each machine, sequence the jobs arbitrarily and do not allow any idle time between jobs.

Step 2. Increase the start time of all jobs by $r_{\max}$, where $r_{\max}=\max_{j=1,2,\ldots,n}\{r_j\}$.

Note that in Step 2, after increasing the start time of all jobs by $r_{\max}$, each job J_j will start processing no earlier than r_j. Thus, algorithm $A1$ always generates a feasible schedule. Let $C_{\max}^{A1}$ denote the makespan of the schedule generated by $A1$, and let $C_{\max}^{*}$ denote the makespan of the optimal schedule. Algorithm $A1$ has a running time of $O(n\log n\log p_{\text{sum}})$ and has a performance guarantee stated in the following theorem.

Theorem 1 $C_{\max}^{A1}/C_{\max}^{*}\leq\frac{7}{3}$.

Proof: Let $\bar{C}_{\max}^{*}$ denote the optimal makespan of the problem when all the job release times are replaced by 0. Clearly, $\bar{C}_{\max}^{*}\leq C_{\max}^{*}$. Let $\bar{C}_{\max}^{A1}$ denote the makespan of the schedule generated by Step 1 of algorithm $A1$. Then, Ou *et al.*'s worst-case bound implies that $\bar{C}_{\max}^{A1}\leq\frac{4}{3}\bar{C}_{\max}^{*}$. Note that $r_{\max}\leq C_{\max}^{*}$. Therefore, $C_{\max}^{A1}=\bar{C}_{\max}^{A1}+r_{\max}\leq\frac{4}{3}\bar{C}_{\max}^{*}+C_{\max}^{*}\leq\frac{7}{3}C_{\max}^{*}$. ∎

Remark 1: In Step 1 of algorithm $A1$, we can also choose to use Ou *et al.*'s $(\frac{4}{3}+\epsilon)$-approximation algorithm. If we do so, $A1$ will have a strongly polynomial running time of $O(n \log n \log \frac{1}{\epsilon})$, and the worst-case error bound will become $C_{\max}^{A1}/C_{\max}^{*} \leq \frac{7}{3}+\epsilon$, where ϵ is a positive constant which may be set arbitrarily close to zero. For example, if we select $\epsilon = \frac{2}{3}$, then this modified version of algorithm $A1$, denoted $A1'$, is a 3-approximation algorithm with a running time of $O(n \log n)$.

Next, we present a more effective approximation algorithm for $P \mid r_j,\ \mathit{incl.\,proc.\,sets} \mid C_{\max}$. This algorithm guarantees that the solution generated has an objective function value no more than twice the optimal solution value. The idea of our algorithm is to search for a feasible schedule via binary search. In each iteration of the search procedure, we attempt to obtain a feasible schedule with makespan no greater than a certain value D by assigning jobs to machines according to the following rules: (a) Jobs are assigned one by one in nonincreasing order of release times. (b) Job J_j can be assigned to a machine M_k (with $k \geq a_j$) only if M_k can process J_j, as well as all its existing jobs, within the time interval $[0, D]$. (c) Among those machines that J_j can be assigned to, we select the least flexible machine (i.e., machine M_k with the smallest possible k).

Let $L = \max_{j=1,2,\ldots,n}\{r_j + p_j\}$ and $U = r_{\max} + p_{\text{sum}}$. Clearly, L and U are lower and upper bounds, respectively, on the optimal solution value of $P \mid r_j,\ \mathit{incl.\,proc.\,sets} \mid C_{\max}$. The approximation algorithm is described formally as follows:

Algorithm $A2$:

Step 1. Re-index the jobs $J_1, J_2, \ldots, J_n$ in such a way that $r_1 \leq r_2 \leq \cdots \leq r_n$. Set $C' \leftarrow L-1$, $C'' \leftarrow U$, $C \leftarrow \lceil (C'+C'')/2 \rceil$, and $D \leftarrow 2C$.

Step 2. (i) For $i = 1, 2, \ldots, m$, set $P_i \leftarrow 0$. For $j = n, n-1, \ldots, 1$, attempt to assign J_j as follows: If there exists machine index ℓ such that $a_j \leq \ell \leq m$ and $r_j + p_j \leq D - P_\ell$, then assign J_j to machine M_k and set $P_k \leftarrow P_k + p_j$, where $k = \min\{\ell \mid a_j \leq \ell \text{ and } r_j + p_j \leq D - P_\ell\}$. Otherwise, we fail to assign all jobs to the machines; stop and go to (ii).

(ii) If we fail to assign all jobs to the machines in (i), then set $C' \leftarrow C$, $C \leftarrow \lceil (C+C'')/2 \rceil$, and $D \leftarrow 2C$. Otherwise, we have a feasible assignment in (i); in such a case, we set

$C'' \leftarrow C$, $C \leftarrow \lceil (C + C')/2 \rceil$, and $D \leftarrow 2C$.

(iii) If $C < C''$, then go to (i).

Step 3. Select the last feasible job assignment obtained in Step 2. On each machine, sequence the jobs in nondecreasing order of r_j. Process each job as early as possible on its assigned machine.

Step 1 of algorithm $A2$ is the initialization step. In each iteration of Step 2, the algorithm tests if all jobs can get assigned to the m machines (with no job completed later than $D = 2C$) following the abovementioned rules (a)–(c). By Lemma 2, a job J_j can be assigned to M_ℓ with $\ell \geq a_j$ if and only if $r_j + p_j \leq D - P_\ell$, where P_ℓ is the total processing time of the existing jobs assigned to that machine. Hence, in Step 2(i) we determine whether J_j can be assigned to M_ℓ by checking the conditions "$r_j + p_j \leq D - P_\ell$" and "$a_j \leq \ell$." The binary search procedure returns a feasible job assignment corresponding to a certain value of C. Thus, if we decrease this value of C by 1, then Step 2(i) will fail to assign all jobs to the machines. In Step 3, given a feasible job assignment, we can determine the job schedule by arranging the jobs on each machine in nondecreasing order of release times (see Lemma 1).

The running time of Steps 1 and 3 is dominated by the binary search. The number of iterations in the binary search is bounded from above by $O(\log U) = O(\log(r_{\max} + p_{\text{sum}}))$. In Step 2, for a given integer D, it takes $O(nm)$ time to determine a feasible assignment (or to confirm that it fails to assign all jobs to the machines). Hence, the overall running time of algorithm $A2$ is $O(nm \log(r_{\max} + p_{\text{sum}}))$, which is polynomial in the input size of the problem.

Let $C_{\max}^{A2}$ denote the makespan of the schedule generated by algorithm $A2$. The following theorem provides us with a performance guarantee on algorithm $A2$.

Theorem 2 $C_{\max}^{A2}/C_{\max}^{*} \leq 2$.

Proof: Suppose, to the contrary, that there exists a problem instance in which $C_{\max}^{A2}/C_{\max}^{*} > 2$. Since $C_{\max}^{A2}$ is the makespan of the schedule generated by algorithm $A2$, Step 2(i) of algorithm $A2$ should fail to assign all jobs to the machines if C is selected in such a way that $D < C_{\max}^{A2}$ (i.e.,

$2C \le C^{A2}_{\max}-1$). We consider the execution of Step 2(i) when $C = \left\lfloor \frac{1}{2}(C^{A2}_{\max}-1)\right\rfloor$, and let J_v denote the first job that fails to get assigned to any machine.

Let $S' = \{J_{v+1}, J_{v+2}, \ldots, J_n\}$, which is the subset of jobs that are assigned to the machines prior to the assignment of J_v. Let $B_i = \{J_j \in S' \mid J_j \text{ is assigned to } M_i\}$ and $S'_i = \{J_j \in S' \mid a_j = i\}$ for $i = 1, 2, \ldots, m$. Define

$$\lambda = \max\{i \mid a_j \ge i \text{ for all } J_j \in B_i \cup B_{i+1} \cup \cdots \cup B_m\}.$$

Thus, all the jobs assigned to machines $M_\lambda, M_{\lambda+1}, \ldots, M_m$ prior to the assignment of J_v have machine indices no smaller than λ. However, for any $\mu = \lambda+1, \lambda+2, \ldots, m$, at least one job assigned to machines $M_\mu, M_{\mu+1}, \ldots, M_m$ has a machine index smaller than μ. Note that the jobs in $B_\lambda \cup B_{\lambda+1} \cup \cdots \cup B_m$ can only be assigned to machines $M_\lambda, M_{\lambda+1}, \ldots, M_m$. Assigning all the jobs in $B_\lambda \cup B_{\lambda+1} \cup \cdots \cup B_m$ to machines $M_\lambda, M_{\lambda+1}, \ldots, M_m$ is possible only if $\frac{1}{m-\lambda+1}\sum_{i=\lambda}^{m}\sum_{J_j \in B_i} p_j \le C^*_{\max}$, which implies that

$$\min_{i=\lambda,\lambda+1,\ldots,m} \left\{ \textstyle\sum_{J_j \in B_i} p_j \right\} \le C^*_{\max}. \tag{1}$$

We now divide the analysis into two cases.

Case 1: $\lambda \ge a_v$. In this case, J_v has a machine index no greater than λ but cannot get assigned to any of $M_\lambda, M_{\lambda+1}, \ldots, M_m$. Thus, by Lemma 2, $r_v + p_v > 2C - \sum_{J_j \in B_i} p_j$ for $i = \lambda, \lambda+1, \ldots, m$, or equivalently, $2C < (r_v + p_v) + \min_{i=\lambda,\lambda+1,\ldots,m}\left\{\sum_{J_j \in B_i} p_j\right\}$. Because $C^*_{\max} \ge r_v + p_v$, we have $2C < C^*_{\max} + \min_{i=\lambda,\lambda+1,\ldots,m}\left\{\sum_{J_j \in B_i} p_j\right\}$.

Case 2: $\lambda < a_v$. In this case, J_v is the first job which cannot get assigned to any of $M_{a_v}, M_{a_v+1}, \ldots, M_m$. Thus, by Lemma 2, $r_v + p_v > 2C - \sum_{J_j \in B_i} p_j$ for $i = a_v, a_v+1, \ldots, m$, or equivalently, $2C < (r_v + p_v) + \min_{i=a_v,a_v+1,\ldots,m}\left\{\sum_{J_j \in B_i} p_j\right\}$. Since $C^*_{\max} \ge r_v + p_v$, we have

$$2C < C^*_{\max} + \min_{i=a_v,a_v+1,\ldots,m} \left\{ \textstyle\sum_{J_j \in B_i} p_j \right\}. \tag{2}$$

By definition of λ, for every $i = \lambda, \lambda+1, \ldots, a_v-1$, there exist $\ell \in \{i+1, i+2, \ldots, m\}$ and a job $J_k \in B_\ell$ such that $a_k \le i$ (otherwise, $a_j \ge i+1$ for all $J_j \in B_{i+1} \cup B_{i+2} \cup \cdots \cup B_m$). Note that $(r_k + p_k) > 2C - \sum_{J_j \in \bar{B}_i} p_j$, where $\bar{B}_i$ is the set of jobs that have been assigned to M_i before the assignment of J_k takes place, since otherwise J_k would be assigned to one of $M_{a_k}, M_{a_k+1}, \ldots, M_i$

instead of M_ℓ. Note also that $C^*_{\max} \geq r_k + p_k$. Thus, $C^*_{\max} > 2C - \sum_{J_j \in \bar{B}_i} p_j$, which implies that $C^*_{\max} > 2C - \sum_{J_j \in B_i} p_j$. Hence,

$$2C < C^*_{\max} + \min_{i=\lambda,\lambda+1,\ldots,a_v-1} \left\{ \sum_{J_j \in B_i} p_j \right\}. \tag{3}$$

Combining (2) and (3), we have $2C < C^*_{\max} + \min_{i=\lambda,\lambda+1,\ldots,m} \left\{ \sum_{J_j \in B_i} p_j \right\}$.

In both Cases 1 and 2, we have $2C < C^*_{\max} + \min_{i=\lambda,\lambda+1,\ldots,m} \left\{ \sum_{J_j \in B_i} p_j \right\}$. By (1), this implies that $2C < 2C^*_{\max}$. Therefore, $C < C^*_{\max}$; that is, $\lfloor \frac{1}{2}(C^{A2}_{\max} - 1) \rfloor < C^*_{\max}$. Because $C^{A2}_{\max}$ and $C^*_{\max}$ are integers, this inequality implies that $C^{A2}_{\max} \leq 2C^*_{\max}$, which is a contradiction. This completes the proof of the theorem. ∎

Remark 2: The worst-case error bound presented in Theorem 2 is asymptotically tight as $m \to \infty$. To see this, consider an example with $n = 2m$, $p_1 = p_2 = \cdots = p_m = m$, $p_{m+1} = p_{m+2} = \cdots = p_{2m} = 1$, $r_1 = r_2 = \cdots = r_m = 0$, $r_{m+1} = r_{m+2} = \cdots = r_{2m} = 1$, $a_i = i$ for $i = 1, 2, \ldots, m$, and $a_{m+1} = a_{m+2} = \cdots = a_{2m} = 1$. It is easy to check that when $C = m - 1$, Step 2(i) of algorithm $A2$ fails to assign all $2m$ jobs to $M_1, M_2, \ldots, M_m$. When $C = m$ (i.e., $D = 2m$), Step 2(i) of $A2$ can assign all jobs to the machines, and the corresponding schedule is depicted in Figure 1(a). Thus, $C^{A2}_{\max} = 2m$. An optimal schedule to this problem instance, which has a makespan of $C^*_{\max} = m+1$, is shown in Figure 1(b). Hence, $C^{A2}_{\max}/C^*_{\max} = 2m/(m+1) \to 2$ as $m \to \infty$.

Remark 3: The running time of algorithm $A2$ is not strongly polynomial. A strongly polynomial time approximation algorithm can be obtained by modifying $A2$ slightly using the method presented in [23]. First, as mentioned in Remark 1, we can use algorithm $A1'$ to obtain a 3-approximation solution to problem $P \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$ in $O(n \log n)$ time. So, instead of using $L = \max_{j=1,2,\ldots,n}\{r_j + p_j\}$ and $U = r_{\max} + p_{\text{sum}}$, we let U be the makespan of this 3-approximation solution and let $L = \frac{1}{3}U$. Then, $L \leq C^*_{\max} \leq U$. Next, in algorithm $A2$, instead of applying binary search on the integer set $\{L, L+1, \ldots, U\}$, we divide the interval $[L, U]$ into K subintervals: $[L, \xi L], (\xi L, \xi^2 L], \ldots, (\xi^{K-2}L, \xi^{K-1}L], (\xi^{K-1}L, U]$, where $\xi = 1 + \frac{\epsilon'}{2}$, $K = \lceil \log_\xi 3 \rceil$, and ϵ' is a prespecified positive constant. We use binary search to search these subintervals. For each subinterval $(C^{(u-1)}, C^{(u)}]$ (or $[C^{(u-1)}, C^{(u)}]$) involved in the binary search, we apply Step 2(i) of algorithm $A2$ with $D = 2C^{(u)}$. This modified version of algorithm $A2$ is a $(2+\epsilon')$-approximation algorithm, and ϵ' may be set arbitrarily close to zero. The binary search procedure takes $O(\log K)$ iterations. It is easy to check that $O(K) \leq O(\frac{1}{\epsilon'})$. Therefore, the running time of the modified algorithm is $O(n \log n + nm \log \frac{1}{\epsilon'})$.

3 A Polynomial Time Approximation Scheme

In this section we develop a PTAS for problem $P \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$. As mentioned in Section 1, Mastrolilli [21] has developed a PTAS for $P \,|\, r_j \,|\, C_{\max}$. We will make use of Mastrolilli's technique of merging small jobs, but we carefully extend his method so that it can be applied to machines with inclusive processing set restrictions. The major differences between our PTAS and Mastrolilli's PTAS are as follows: (i) Mastrolilli categorizes the jobs based on their release times, while we categorizes them based on their release times as well as their machine indices. (ii) When we assign the small jobs to machines, we assign them according to their machine indices. (iii) Mastrolilli uses an integer linear program (ILP) to generate a schedule after rounding the job processing times, while we use a dynamic program to do so (see Remark 6 below).

In our PTAS, we first apply algorithm $A2$ to obtain a 2-approximation solution to the given problem instance, and let UB denote the makespan of the schedule obtained. Then, UB/2 is a lower bound on $C^*_{\max}$. Note that $r_{\max} + 1$ and $p_{\max}$ are also lower bounds on $C^*_{\max}$, where $r_{\max} = \max_{j=1,2,\ldots,n}\{r_j\}$ and $p_{\max} = \max_{j=1,2,\ldots,n}\{p_j\}$. Let $\text{LB} = \max\{r_{\max}+1, p_{\max}, \text{UB}/2\}$. We have $\text{LB} \leq C^*_{\max} \leq 2\text{LB}$. Next, we divide each release time and processing time by LB. Then, $r_{\max} < 1$, $p_{\max} \leq 1$, and

$$1 \leq C^*_{\max} \leq 2. \tag{4}$$

Let $\bar{\epsilon}$ be an arbitrary small rational number, where $0 < \bar{\epsilon} < 1$. For simplicity, we assume that $1/\bar{\epsilon}$ is integral. (For a given $\bar{\epsilon}$, if $1/\bar{\epsilon}$ is not integral, then we replace $\bar{\epsilon}$ by $\epsilon' = \frac{1}{\lceil 1/\bar{\epsilon} \rceil}$. We have $\epsilon' < \bar{\epsilon}$ and $O(1/\epsilon') = O(1/\bar{\epsilon})$. Thus, replacing $\bar{\epsilon}$ by ϵ' does not affect the validity of our PTAS.)

We consider a "release time rounding procedure." In this procedure, we round every release time down to the nearest multiple of $\bar{\epsilon}$, obtain an approximation solution to the problem with rounded release times (using a method described later), and then add $\bar{\epsilon}$ to each job's start time. Clearly, this procedure will generate a feasible schedule (i.e., a schedule in which every job starts no earlier than its release time). Let σ_1^* denote the schedule generated by this procedure when we solve the rounded release time problem optimally. We have the following lemma:

Lemma 3 $C_{\max}^* \leq C_{\max}(\sigma_1^*) \leq (1+\bar{\epsilon})C_{\max}^*$.

Proof: The inequality "$C_{\max}^* \leq C_{\max}(\sigma_1^*)$" is obvious. Note that after rounding every release time down and solving the problem optimally, the makespan of the solution is no greater than $C_{\max}^*$. When we add $\bar{\epsilon}$ to each job's start time, the makespan of the schedule increases by no more than $\bar{\epsilon}$. Thus, $C_{\max}(\sigma_1^*) \leq C_{\max}^* + \bar{\epsilon} \leq (1+\bar{\epsilon})C_{\max}^*$. ∎

We now focus on problem instances with rounded release times and discuss how to obtain approximation solutions to those instances. Since $r_{\max} < 1$, the number of different release times is bounded from above by $1/\bar{\epsilon}$. We refer to those jobs with processing times less than $\bar{\epsilon}^2$ as "small jobs" and the other jobs as "big jobs." Let h be the number of distinct release times in the problem instance, where $h \leq 1/\bar{\epsilon}$. Let $r^{(1)}, r^{(2)}, \ldots, r^{(h)}$ be those release times. We refer to a job with release time $r^{(k)}$ as a "type-k job" ($k = 1, 2, \ldots, h$). Recall that the job set $\mathcal{J}$ is partitioned into $S_1, S_2, \ldots, S_m$ according to the machine indices of the jobs. We now further partition each S_i into subsets $S_i^{(1)}, S_i^{(2)}, \ldots, S_i^{(h)}$, where the jobs in $S_i^{(k)}$ are type-k jobs with machine index i ($k = 1, 2, \ldots, h$; $i = 1, 2, \ldots, m$).

To obtain an approximation solution to a problem instance with rounded release times, we use the following "job merging procedure": For each i and k, let J_a and J_b be any two small jobs in $S_i^{(k)}$. We merge these two small jobs to form a composed job J_c such that J_c has the same release time as J_a (and J_b), and that $p_c = p_a + p_b$ (see [21]). In other words, we require J_a and J_b to be processed together on the same machine one immediately after the other. We repeat this merging process until each subset $S_i^{(k)}$ contains at most one small job. Then, we obtain an approximation solution to the resulting problem instance (using a method described later).

We denote the subset $S_i^{(k)}$ after the merging process as $\bar{S}_i^{(k)}$. Clearly, the processing time of a composed job is less than $2\bar{\epsilon}^2$, and there is at most one small job in each $\bar{S}_i^{(k)}$. Let σ_2^* denote the schedule generated by the job merging procedure when we solve the resulting problem instance optimally.

Lemma 4 $C_{\max}(\sigma_1^*) \leq C_{\max}(\sigma_2^*) \leq (1 + 2\bar{\epsilon})C_{\max}(\sigma_1^*)$.

Proof: The inequality "$C_{\max}(\sigma_1^*) \leq C_{\max}(\sigma_2^*)$" is obvious. To prove the lemma, it suffices to show that there exists a feasible schedule σ_2 for the instance obtained from the job merging process such that $C_{\max}(\sigma_2) \leq (1 + 2\bar{\epsilon})C_{\max}(\sigma_1^*)$. For simplicity, we assume that in schedule σ_1^* the jobs on each machine are sequenced in nondecreasing order of release times (see Lemma 1). Let $A_i^{(k)}$ denote the set of type-k small jobs that are processed by M_i in schedule σ_1^* ($k = 1, 2, \ldots, h$; $i = 1, 2, \ldots, m$). Let $\mathcal{B}$ denote the set of non-composed big jobs after the job merging process, and $\mathcal{A} = \mathcal{J} \setminus \mathcal{B}$ denote the set of composed jobs and small jobs.

We construct σ_2 as follows: (i) For each $k = 1, 2, \ldots, h$, we assign the jobs in $\mathcal{A} \cap (\bigcup_{i=1}^m \bar{S}_i^{(k)})$ one by one to the machines, starting from those jobs with the smallest machine index. We first assign them to M_1 until either the total processing time of the assigned type-k jobs on M_1 exceeds $\sum_{J_j \in A_1^{(k)}} p_j$ or there is no more unassigned job in $\mathcal{A} \cap \bar{S}_1^{(k)}$. We then assign them to M_2 until either the total processing time of the assigned type-k jobs on M_2 exceeds $\sum_{J_j \in A_2^{(k)}} p_j$ or there is no more unassigned job in $\mathcal{A} \cap (\bar{S}_1^{(k)} \cup \bar{S}_2^{(k)})$. Next, we assign them to M_3 until either the total processing time of the assigned type-k jobs on M_3 exceeds $\sum_{J_j \in A_3^{(k)}} p_j$ or there is no more unassigned job in $\mathcal{A} \cap (\bar{S}_1^{(k)} \cup \bar{S}_2^{(k)} \cup \bar{S}_3^{(k)})$, and so on. (ii) For each $J_j \in \mathcal{B}$, we assign J_j to the machine which processes J_j in schedule σ_1^*. (iii) On each machine, we sequence the jobs in nondecreasing order of release times, and schedule each job to start as soon as the job has been released and the machine has completed the previous job.

Note that in Step (i), when we assign type-k jobs to a machine M_i, we always assign enough type-k jobs so that their total processing time is greater than the total type-k job processing time in $A_i^{(k)}$, unless we run out of type-k jobs that can be processed by M_i. Hence, all type-k jobs must get assigned to the m machines.

On the other hand, the total processing time of the composed type-k jobs and small type-k jobs assigned to M_i in Step (i) cannot exceed the total processing time of the small type-k jobs on M_i in σ_1^* by more than $2\bar{\epsilon}^2$ (because each composed job has a processing time less than $2\bar{\epsilon}^2$). Hence, the completion time of the last job on M_i in σ_2 cannot exceed the completion time of the last job on M_i in σ_1^* by more than $2h\bar{\epsilon}^2$ ($i = 1, 2, \ldots, m$). This implies that $C_{\max}(\sigma_2) \leq C_{\max}(\sigma_1^*) + 2h\bar{\epsilon}^2 \leq C_{\max}(\sigma_1^*) + 2\bar{\epsilon}$. Since $C_{\max}(\sigma_1^*) \geq C^*_{\max} \geq 1$, we have $C_{\max}(\sigma_2^*) \leq (1 + 2\bar{\epsilon})C_{\max}(\sigma_1^*)$. ∎

We now focus on problem instances resulted from the job merging process and discuss how to obtain approximation solutions to those instances. We consider the following "processing time rounding procedure": First, for each big job J_j, let $y_j = \max\left\{\ell \;\middle|\; \bar{\epsilon}^2(1+\bar{\epsilon})^\ell \leq p_j;\ \ell = 0, 1, 2, \ldots\right\}$. Then, $\bar{\epsilon}^2(1+\bar{\epsilon})^{y_j} \leq p_j < \bar{\epsilon}^2(1+\bar{\epsilon})^{y_j+1}$, and we round p_j down to $\bar{\epsilon}^2(1+\bar{\epsilon})^{y_j}$. Next, we ignore the small jobs and obtain an optimal solution to the problem instance with rounded processing times (using a method described later). Then, we restore the big jobs' original processing times and replace the composed jobs by their original small jobs. After that, we assign each remaining small job J_j to machine M_{a_j}. Finally, on each machine, we sequence the jobs in nondecreasing order of release times, and schedule each job to start as soon as the job has been released and the machine has completed the previous job.

Let σ_3^* denote the schedule generated by the processing time rounding procedure.

Lemma 5 $C_{\max}(\sigma_2^*) \leq C_{\max}(\sigma_3^*) \leq (1 + 2\bar{\epsilon})C_{\max}(\sigma_2^*)$.

Proof: The inequality "$C_{\max}(\sigma_2^*) \leq C_{\max}(\sigma_3^*)$" is obvious. Note that after rounding the processing times down and solving the problem optimally, the makespan of the solution is no greater than $C_{\max}(\sigma_2^*)$. When we restore the original processing times of the big jobs, the makespan of the schedule increases by a factor of no more than $1 + \bar{\epsilon}$ (i.e., increases by an absolute amount of no more than $\bar{\epsilon}C_{\max}(\sigma_2^*)$). Since there is at most one small job in each $\bar{S}_i^{(k)}$, the number of small jobs in $\bar{S}_i^{(1)} \cup \bar{S}_i^{(2)} \cup \cdots \cup \bar{S}_i^{(h)}$ is at most $1/\bar{\epsilon}$. Hence, inserting the remaining small jobs into the schedule increases the makespan by no more than $(1/\bar{\epsilon}) \cdot \bar{\epsilon}^2 = \bar{\epsilon} \leq \bar{\epsilon}C_{\max}(\sigma_2^*)$. Therefore, $C_{\max}(\sigma_3^*) \leq (1 + 2\bar{\epsilon})C_{\max}(\sigma_2^*)$. ∎

Finally, we describe a dynamic program for obtaining an optimal solution to the problem with rounded processing times. We first categorize the jobs in such a way that two jobs belong to the same category if they have the same release time and the same processing time. Let τ denote the number of job categories.

Lemma 6 $\tau \leq (1/\bar{\epsilon})\lfloor 1+\log_{1+\bar{\epsilon}}(1/\bar{\epsilon}^2)\rfloor$.

Proof: Since $p_{\max} \leq 1$, each job processing time $\bar{\epsilon}^2(1+\bar{\epsilon})^{y_j}$ is at most 1, which implies that $y_j \leq \log_{1+\bar{\epsilon}}(1/\bar{\epsilon}^2)$. Thus, the number of possible values of y_j is no more than $\lfloor 1+\log_{1+\bar{\epsilon}}(1/\bar{\epsilon}^2)\rfloor$. The number of distinct job release times is no more than $1/\bar{\epsilon}$. Therefore, the number of job categories is no more than $(1/\bar{\epsilon})\lfloor 1+\log_{1+\bar{\epsilon}}(1/\bar{\epsilon}^2)\rfloor$. ∎

We index the job categories as $1,2,\ldots,\tau$. Let n_ℓ denote the number of jobs in category ℓ ($\ell=1,2,\ldots,\tau$). Let $n_\ell(i)$ denote the number of jobs with machine index i in category ℓ. Thus, $\sum_{i=1}^m n_\ell(i)=n_\ell$. Let $n_{\ell i}=\sum_{k=1}^i n_\ell(k)$, which is the maximum number of jobs from category ℓ that can be assigned to machine M_i ($i=1,2,\ldots,m$). Let $x_{\ell i}$ be a decision variable representing the number of jobs from category ℓ that are assigned to M_i. Clearly, we have a constraint of "$x_{\ell i} \leq n_{\ell i}$" for $\ell=1,2,\ldots,\tau$ and $i=1,2,\ldots,m$. Hence, we call $(x_{1i},x_{2i},\ldots,x_{\tau i})$ a "feasible assignment" for M_i if $x_{\ell i} \leq n_{\ell i}$ for $\ell=1,2,\ldots,\tau$. Let

$$X_i = \big\{(x_{1i},x_{2i},\ldots,x_{\tau i}) \,\big|\, x_{\ell i}=0,1,\ldots,n_{\ell i} \text{ for } \ell=1,2,\ldots,\tau\big\},$$

which is the set of all feasible assignments for M_i. By Lemma 1, we may assume that the assigned jobs are processed in nondecreasing order of their release times and are processed as early as possible. Let $C(x_{1i},x_{2i},\ldots,x_{\tau i})$ denote the completion time of the last job on M_i if the feasible assignment $(x_{1i},x_{2i},\ldots,x_{\tau i})$ is adopted. It is easy to see that for each $(x_{1i},x_{2i},\ldots,x_{\tau i}) \in X_i$, $C(x_{1i},x_{2i},\ldots,x_{\tau i})$ can be determined in $O(\tau)$ time if the job categories are indexed in nondecreasing order of job release times. Note that $X_1 \subseteq X_2 \subseteq \cdots \subseteq X_m$ and $|X_m|=\prod_{\ell=1}^{\tau}(n_\ell+1) \leq O(n^\tau)$.

Let $Y_i = \big\{(x_{1i},x_{2i},\ldots,x_{\tau i}) \in X_i \mid C(x_{1i},x_{2i},\ldots,x_{\tau i}) \leq 2\big\}$ for $i=1,2,\ldots,m$. Since the processing time of each job is no less than $\bar{\epsilon}^2$, an assignment $(x_{1i},x_{2i},\ldots,x_{\tau i})$ in Y_i consists of at most $2/\bar{\epsilon}^2$ jobs. Thus, $|Y_i| \leq (\tau+1)^{2/\bar{\epsilon}^2}$ (to see this inequality, consider $2/\bar{\epsilon}^2$ job positions, where each position may either be occupied by a job of any of the τ categories or remain empty). From (4), $C^*_{\max} \le 2$. Hence, it suffices to consider assignments in Y_i when we select feasible assignments for M_i.

Define $F_i(v_1, v_2, \ldots, v_\tau)$ as the minimum possible makespan if we schedule v_ℓ jobs of category ℓ (for $\ell = 1, 2, \ldots, \tau$) to machines $M_1, M_2, \ldots, M_i$, subject to the constraint that only assignments in Y_k can be used for M_k (for $k = 1, 2, \ldots, i$). We have the following recurrence relation:

$$F_i(v_1, v_2, \ldots, v_\tau) = \min_{\substack{(x_1,x_2,\ldots,x_\tau)\in Y_i \text{ s.t.} \\ (x_1,x_2,\ldots,x_\tau)\le(v_1,v_2,\ldots,v_\tau)}} \Big\{\max\big\{C(x_1, x_2, \ldots, x_\tau), F_{i-1}(v_1-x_1, v_2-x_2, \ldots, v_\tau-x_\tau)\big\}\Big\}$$

for $i = 2, 3, \ldots, m$ and $(v_1, v_2, \ldots, v_\tau) \in X_i$. The boundary conditions are:

$$F_1(v_1, v_2, \ldots, v_\tau) = \begin{cases} C(v_1, v_2, \ldots, v_\tau), & \text{if } (v_1, v_2, \ldots, v_\tau) \in Y_1; \\ +\infty, & \text{otherwise;} \end{cases}$$

and

$$F_i(v_1, v_2, \ldots, v_\tau) = +\infty \text{ if } (v_1, v_2, \ldots, v_\tau) \notin X_i \quad (i = 2, 3, \ldots, m).$$

The makespan of the optimal schedule is given as $F_m(n_1, n_2, \ldots, n_\tau)$.

Indexing the job categories in nondecreasing order of job release times takes $O(\tau \log \tau)$ time. Predetermining the values of $C(x_{1i}, x_{2i}, \ldots, x_{\tau i})$ for all $(x_{1i}, x_{2i}, \ldots, x_{\tau i}) \in Y_i$ and all $i = 1, 2, \ldots, m$ takes $O(\tau|Y_m|)$ time. Executing the above dynamic program takes $O(m|X_m||Y_m|)$ time. Thus, the overall running time required for obtaining an optimal solution to the problem with rounded processing times is $O\big(n^\tau m(\tau + 1)^{2/\bar{\epsilon}^2}\big)$. Hence, the running time of the overall solution procedure, including the determination of LB, the release time rounding process, the job merging process, the processing time rounding process, and the above dynamic programming procedure, is $O\big(nm \log(r_{\max} + p_{\text{sum}}) + n^\tau m(\tau+1)^{2/\bar{\epsilon}^2}\big)$. Note that by Lemma 6, τ is a constant for fixed $\bar{\epsilon}$. This result is summarized in the following theorem:

Theorem 3 *Problem $P \,|\, r_j$, incl. proc. sets $|\, C_{\max}$ admits a PTAS.*

Remark 4: In the above PTAS, the computational time needed for determining LB is $O\big(nm \log(r_{\max} + p_{\text{sum}})\big)$. An alternative way to determine a lower bound on $C^*_{\max}$ is to use the strongly polynomial time $(2+\epsilon')$-approximation algorithm described in Remark 3. Let UB' denote the makespan of the schedule obtained by that algorithm, and let $\text{LB}' = \max\{r_{\max}+1, p_{\max}, \text{UB}'/(2+\epsilon')\}$. Then, $\text{LB}' \le C^*_{\max} \le (2+\epsilon')\text{LB}'$. If we use LB' instead of LB in the above PTAS development, then the running time of the PTAS becomes $O\big(nm\log\frac{1}{\epsilon'}+n^{\tau}m(\tau+1)^{(2+\epsilon')/\bar{\epsilon}^2}\big)$.

Remark 5: Another way to determine a lower bound on $C^*_{\max}$ is as follows: Note that the jobs in $S_m \cup S_{m-1} \cup \cdots \cup S_{m-k+1}$ must be processed by machines $M_{m-k+1}, M_{m-k+2}, \ldots, M_m$. Thus, $C^*_{\max} \ge \frac{1}{k}\sum_{J_j \in S_m \cup S_{m-1} \cup \cdots \cup S_{m-k+1}} p_j$ for $k = 1, 2, \ldots, m$. Let

$$L = \max_{k=1,2,\ldots,m}\left\{\frac{1}{k}\sum_{J_j \in S_m \cup S_{m-1} \cup \cdots \cup S_{m-k+1}} p_j\right\}$$

and $\text{LB}'' = \max\{r_{\max}+1, p_{\max}, L\}$. Then, $\text{LB}'' \le C^*_{\max}$. Determining LB'' requires $O(n)$ time. Note that a feasible solution to the problem with a makespan guaranteed no more than $3\text{LB}''$ can be constructed as follows: (i) Replace all job release times by 0. (ii) For $k = m, m-1, \ldots, 1$, assign the jobs in S_k one by one to machines $M_k, M_{k+1}, \ldots, M_m$; every time a job is assigned, it is put on a machine with the minimal current workload. (iii) Increase the start time of all jobs by $r_{\max}$. Using the same argument as in Section 9.0 of [17], it is easy to show that the makespan of the schedule generated by steps (i)–(iii) must be no more than $L + p_{\max} + r_{\max} \le 3\text{LB}''$. Hence, $\text{LB}'' \le C^*_{\max} \le 3\text{LB}''$. If we use LB'' instead of LB in the above PTAS development, then the running time of the PTAS becomes $O(n^{\tau}m(\tau+1)^{3/\bar{\epsilon}^2})$.

Remark 6: As mentioned earlier, Mastrolilli [21] has developed a PTAS for problem $P\,|\,r_j\,|\,L_{\max}$. His PTAS is "efficient" in the sense that it can generate a $(1+\epsilon)$-approximation solution in $O(n+\bar{f}(\epsilon))$ time, where $\bar{f}(\epsilon)$ is a constant for fixed ϵ. Such a computational complexity is achieved by solving the rounded processing time problem as an ILP with a constant number of variables and a constant number of constraints using Lenstra's [18] algorithm. For our problem with inclusive processing sets, if we formulate the rounded processing time problem as an ILP, the number of constraints will depend on m, and if we solve the ILP using Lenstra's algorithm, the computational complexity will be very high.

4 When the Number of Machines Is Fixed

In this section we develop an FPTAS for problem $Pm \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$. First, we solve the given instance of $Pm \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$ using algorithm $A1'$ (see Remark 1), and let UB be the makespan of the schedule obtained. Then, $C^*_{\max} \leq \mathrm{UB}$. Note that $Pm \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$ is a special case of $Rm \,|\, r_j \,|\, C_{\max}$. In problem $Rm \,|\, r_j \,|\, C_{\max}$, each job J_j has a nonnegative release time r_j, and it has a processing time p_{ij} if it is assigned to machine M_i. Given a problem instance of $Pm \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$, we can convert it into an instance of $Rm \,|\, r_j \,|\, C_{\max}$ by defining $p_{ij} = p_j$ if $i \geq a_j$, and $p_{ij} = \mathrm{UB} + 1$ if $i < a_j$, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. Note that $Rm \,|\, r_j \,|\, C_{\max}$ is a generalization $Rm \,|\,|\, C_{\max}$, which is known to be NP-hard [17]. However, to the best of our knowledge, no pseudo-polynomial time algorithm has been developed for $Rm \,|\, r_j \,|\, C_{\max}$.

Next, we present a dynamic program which can determine an optimal solution to $Rm \,|\, r_j \,|\, C_{\max}$ in pseudo-polynomial time. Let $\bar{U}$ be any upper bound on the makespan of the optimal schedule of $Rm \,|\, r_j \,|\, C_{\max}$ (e.g., if the problem is converted from an instance of $Pm \,|\, r_j,\, incl.\, proc.\, sets \,|\, C_{\max}$, then we may set $\bar{U} = \mathrm{UB}$). We re-index the jobs in such a way that $r_1 \leq r_2 \leq \cdots \leq r_n$. Define

$$G(j; x_1, x_2, \ldots, x_m) = \begin{cases} 1, & \text{if } J_1, J_2, \ldots, J_j \text{ can be scheduled on } M_1, M_2, \ldots, M_m \text{ such} \\ & \text{that the makespan of } M_i \text{ is no more than } x_i \ (i = 1, 2, \ldots, m); \\ 0, & \text{otherwise;} \end{cases}$$

for $j = 0, 1, \ldots, n$ and any nonnegative integer x_i $(i = 1, 2, \ldots, m)$. The recurrence relation is

$$G(j; x_1, x_2, \ldots, x_m) = \max_{\substack{i=1,2,\ldots,m \\ \text{s.t. } x_i \geq r_j + p_{ij}}} \Big\{ G(j-1; x_1, \ldots, x_{i-1}, x_i - p_{ij}, x_{i+1}, \ldots, x_m) \Big\}$$

for all $j = 1, 2, \ldots, n$ and $0 \leq x_i \leq \bar{U}$ $(i = 1, 2, \ldots, m)$ such that $\max_{i=1,2,\ldots,m}\{x_i - r_j - p_{ij}\} \geq 0$. The boundary conditions are

$$G(0; x_1, x_2, \ldots, x_m) = 1 \text{ for all } (x_1, x_2, \ldots, x_m) \geq (0, 0, \ldots, 0),$$

and for $j = 1, 2, \ldots, n$,

$$G(j; x_1, x_2, \ldots, x_m) = 0 \text{ if } x_i < r_j + p_{ij} \text{ for } i = 1, 2, \ldots, m.$$

The optimal makespan of the schedule is given as

$$\min\left\{\max\{x_1, x_2, \ldots, x_m\} \;\middle|\; G(n; x_1, x_2, \ldots, x_m) = 1;\ 0 \leq x_1, x_2, \ldots, x_m \leq \bar{U}\right\}.$$

The running time of this dynamic program is $O(n\bar{U}^m)$, which is pseudo-polynomial when m is fixed.

Let ϵ be a given constant, where $0 < \epsilon < 1$. We now construct a polynomial-time ϵ-approximation algorithm for $Pm \,|\, r_j,\ incl.\, proc.\, sets \,|\, C_{\max}$. Let $\mathrm{LB}' = \mathrm{UB}/3$. Then, by Remark 1, $\mathrm{LB}' \leq C^*_{\max} \leq \mathrm{UB}$. We replace all job release times r_j by $\lfloor \frac{r_j(n+1)}{\epsilon \cdot \mathrm{LB}'} \rfloor \frac{\epsilon \cdot \mathrm{LB}'}{n+1}$ and all job processing times p_{ij} by $\lfloor \frac{p_{ij}(n+1)}{\epsilon \cdot \mathrm{LB}'} \rfloor \frac{\epsilon \cdot \mathrm{LB}'}{n+1}$, and then obtain an optimal schedule σ to the problem with these rounded data. We obtain an approximated solution to the original problem by taking schedule σ and restoring the original release times and processing times. It is easy to see that the makespan of this approximated solution, $C^A_{\max}$, cannot be greater than the makespan of σ by more than $(n+1) \cdot \frac{\epsilon \cdot \mathrm{LB}'}{n+1} = \epsilon \cdot \mathrm{LB}'$. Hence, $C^A_{\max} \leq (1+\epsilon) C^*_{\max}$.

To obtain an optimal schedule σ to the problem with the rounded data, we do the following: Since all release times and processing times are integer multiples of $\frac{\epsilon \cdot \mathrm{LB}'}{n+1}$, we divide these parameters by $\frac{\epsilon \cdot \mathrm{LB}'}{n+1}$ and then apply the above dynamic program (with $\bar{U} = \mathrm{UB} \cdot \frac{n+1}{\epsilon \cdot \mathrm{LB}'}$). Then, we take the dynamic programming solution and multiply all job start times by $\frac{\epsilon \cdot \mathrm{LB}'}{n+1}$. The running time of this dynamic programming procedure is $O(n\bar{U}^m) = O\big(n\big(\mathrm{UB} \cdot \frac{n+1}{\epsilon \cdot \mathrm{LB}'}\big)^m\big) = O\big(n^{m+1}\big(\frac{1}{\epsilon}\big)^m\big)$. The computational time required to obtain UB is $O(n \log n)$, which is dominated by the running time of the dynamic programming procedure. The running time $O\big(n^{m+1}\big(\frac{1}{\epsilon}\big)^m\big)$ is polynomial in both $1/\epsilon$ and the input size of the problem. Therefore, we have the following result:

Theorem 4 *Problem $Pm \,|\, r_j,\ incl.\, proc.\, sets \,|\, C_{\max}$ admits an FPTAS.*

5 Conclusions

We have presented an efficient 2-approximation algorithm for our parallel machine scheduling problem with inclusive set restrictions and job release times. We have also presented a PTAS for the problem, as well as an FPTAS for the case in which the number of machines is fixed.

An interesting future research direction is to investigate other parallel machine scheduling models with inclusive processing sets, job release times, and other objective functions, such as minimizing total (unweighted) job completion times, minimizing total weighted job completion times, and bicriterion objectives. Further extensions to uniform machines are also worth investigating, because in some applications, machines with different processing capabilities may be operated at different speeds.

Acknowledgments

The work of the second author was done in part in the Department of Logistics at The Hong Kong Polytechnic University. This research was supported in part by the Research Grants Council of Hong Kong under grant PolyU5222/06E. The second author was also supported in part by the National Nature Science Foundation of China under grant 70871059.

References

[1] Bar-Noy, A., Freund, A., Naor, J., On-line load balancing in a hierarchical server topology. *SIAM Journal on Computing* 31 (2001) 527–549.

[2] Chen, B., Potts, C.N., Woeginger, G.J., A review of machine scheduling: Complexity, algorithms and approximability. *Handbook of Combinatorial Optimization, Volume 3,* Du, D.-Z., Pardalos, P.M. (Editors), Kluwer Academic Publishers, Boston, 1998, pp. 21–169.

[3] Chekuri, C., Motwani, R., Natarajan, B., Stein, C., Approximation techniques for average completion time scheduling. *SIAM Journal on Computing* 31 (2001) 146–166.

[4] Fishkin, A.V., Jansen, K., Mastrolilli, M., Grouping techniques for scheduling problems: Simpler and faster. *Algorithmica* 51 (2008) 183–199.

[5] Glass, C.A., Kellerer, H., Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics* 54 (2007) 250–257.

[6] Glass, C.A., Mills, H.R., Scheduling unit length jobs with parallel nested machine processing set restrictions. *Computers and Operations Research* 33 (2006) 620–638.

[7] Hall, L.A., Shmoys, D.B., Approximation schemes for constrained scheduling problems. *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science* (1989) 134–139.

[8] Horowitz, E., Sahni, S., Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the Association for Computing Machinery* 23 (1976) 317–327.

[9] Huo, Y., Leung, J.Y.-T., Wang, X., Preemptive scheduling algorithms with nested and inclusive processing set restrictions. Working paper, Department of Computer Science, New Jersey Institute of Technology.

[10] Hwang, H.-C., Chang, S.Y., Lee, K., Parallel machine scheduling under a grade of service provision. *Computers and Operations Research* 31 (2004) 2055–2061.

[11] Jansen, K., Porkolab, L., Improved approximation schemes for scheduling unrelated parallel machines. *Mathematics of Operations Research* 26 (2001) 324–338.

[12] Ji, M., Cheng, T.C.E., An FPTAS for parallel-machine scheduling under a grade of service provision to minimize makespan. *Information Processing Letters* 108 (2008) 171-174.

[13] Jiang, Y., Online scheduling on parallel machines with two GoS levels. *Journal of Combinatorial Optimization* 16 (2008) 28–38.

[14] Jiang, Y.-W., He, Y., Tang, C.-M., Optimal online algorithms for scheduling on two identical machines under a grade of service. *Journal of Zhejiang University SCIENCE A* 7 (2006) 309–314.

[15] Kafura, D.G., Shen, V.Y., Task scheduling on a multiprocessor system with independent memories. *SIAM Journal on Computing* 6 (1977) 167–187.

[16] Lai, T.-H., Sahni, S., Preemptive scheduling of a multiprocessor system with memories to minimize maximum lateness. *SIAM Journal on Computing* 13 (1984) 690–704.

[17] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B., Sequencing and scheduling: Algorithms and complexity. *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory,* Graves, S.C., Rinnooy Kan, A.H.G., Zipkin, P.H. (Editors), North-Holland, Amsterdam, 1993, pp. 445–522.

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

[19] Lenstra, J.K., Shmoys, D.B., Tardos, E., Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming* 46 (1990) 259–271.

[20] Leung, J.Y.-T., Li, C.-L., Scheduling with processing set restrictions: A survey. *International Journal of Production Economics* 116 (2008) 251–262.

[21] Mastrolilli, M., Efficient approximation schemes for scheduling problems with release dates and delivery times. *Journal of Scheduling* 6 (2003) 521–531.

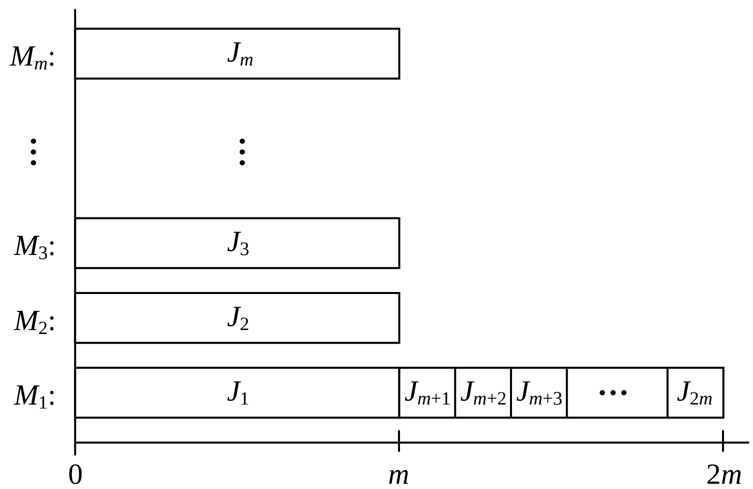
[22] Mastrolilli, M., Scheduling to minimize max flow time: Off-line and on-line algorithms. *International Journal of Foundations of Computer Science* 15 (2004) 385–401.

[23] Ou, J., Leung, J.Y.-T., Li, C.-L., Scheduling parallel machines with inclusive processing set restrictions. *Naval Research Logistics* 55 (2008) 328–338.

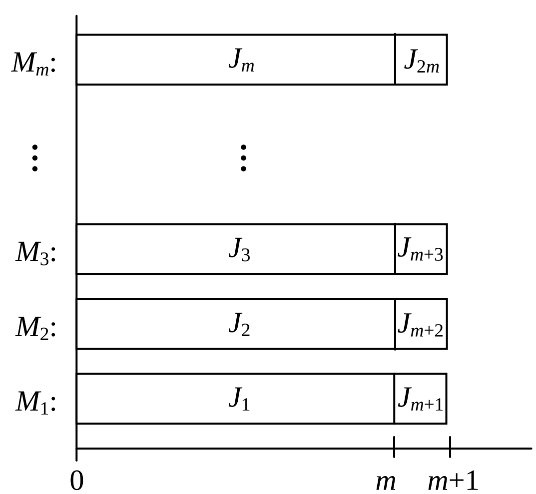
[24] Park, J., Chang, S.Y., Lee, K., Online and semi-online scheduling of two machines under a grade of service provision. *Operations Research Letters* 34 (2006) 692–696.

[25] Schulz, A.S., Skutella, M., Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics* 15 (2002) 450–469.

[26] Shchepin, E.V., Vakhania, N., An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters* 33 (2005) 127–133.



(a) Schedule obtained by Algorithm $A2$



(b) Optimal schedule

Figure 1. A worst-case example.