https://doi.org/10.1016/j.ejor.2017.01.013

Fast approximation algorithms for uniform machine scheduling with processing set restrictions

Joseph Y-T. Leung *

Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102

C. T. Ng

Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Kowloon, Hong Kong

Abstract

We consider the problem of nonpreemptively scheduling a set of independent jobs on a set of uniform machines, where each job has a set of machines to which it can be assigned. This kind of restriction is called the processing set restriction. In the literature there are many kinds of processing set restrictions that have been studied. In this paper we consider two kinds: the "inclusive processing set" and the "tree-hierarchical processing set". Epstein and Levin (2011) have given Polynomial Time Approximation Schemes (PTAS) to solve both classes. However, the running times of their PTAS are rather high. In this paper, we give fast approximation algorithms for both cases and show that they both have a worst-case performance bound of 4/3. Moreover, we show that the bounds are achievable.

Key words: Uniform machines, inclusive processing set, tree-hierarchical processing set, nonpreemptive scheduling, makespan, worst-case bound.

* Corresponding author.
Email addresses: leung@njit.edu (Joseph Y-T. Leung), lgtctng@polyu.edu.hk (C. T. Ng).

1 Introduction

We consider the problem of nonpreemptively scheduling n independent jobs $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ on m uniform machines $\mathcal{M} = \{M_1, M_2, \cdots, M_m\}$, where the machines differ in their speeds but not in their functionality. Each job J_j has a processing requirement p_j and each machine M_i has a speed s_i. If job J_j is processed by machine M_i, then it takes p_j/s_i units of real time to complete. Each job J_j has a set of machines $\mathcal{M}_j \subseteq \mathcal{M}$ to which it can be assigned. Our goal is to find a schedule so that each job J_j is assigned to one of the machines in $\mathcal{M}_j$ and that the makespan $C_{\max}$ is minimized.

The problem we study belongs to a class of machine scheduling problems with processing set restrictions. As stated in the survey paper by Leung and Li (2016), there are several special forms of processing set restrictions studied by researchers, including "inclusive processing sets", "nested processing sets", "interval processing sets", "tree-hierarchical processing sets", and "arbitrary processing sets". In this paper we study the classes of "inclusive processing set" and "tree-hierarchical processing set" restrictions.

The "inclusive processing set" restriction has the property that for each pair of jobs J_i and J_j, either $\mathcal{M}_i \subseteq \mathcal{M}_j$ or $\mathcal{M}_j \subseteq \mathcal{M}_i$. The "tree-hierarchical processing set" has the property that each machine M_k is associated with a vertex of a tree, and that the processing set of a job J_j is the set of machines composed of its associated vertex as well as all the vertices on the unique path from its associated vertex to the root of the tree. Clearly, the "inclusive processing set" restriction is a special case of "tree-hierarchical processing set" restriction since we can view the machines in "inclusive processing set" restriction as forming a chain. The 3-field notation for "inclusive processing set" with uniform machines is $Q \mid \mathcal{M}_j(inclusive) \mid C_{\max}$,

while that of "tree-hierarchical processing set" is $Q \mid \mathcal{M}_j(tree) \mid C_{\max}$. The problem $Q \mid \mathcal{M}_j(inclusive) \mid C_{\max}$ is strongly NP-hard, since a special case of the problem, $Q \mid\mid C_{\max}$, is strongly NP-hard.

The problem $Q \mid \mathcal{M}_j(inclusive) \mid C_{\max}$ has several interesting real-life applications. As noted in the paper by Epstein and Levin (2011), consider the situation where a job must be completed within a certain amount of time once it starts the processing. In this case, each job J_j has a minimum speed level l_j, and the job can only be processed by a machine M_k if $s_k \geq l_j$. That is, the job must be processed by a machine fast enough to complete it within the required time interval. Another application is when we have a set of workers with different skill levels, and a job requires a worker with a minimum skill level to process. A worker with a higher level of expertise can handle a larger set of jobs and is also faster in completing the jobs.

The problem $Q \mid \mathcal{M}_j(tree) \mid C_{\max}$ also has real-life applications. Consider a company with the following management structure. There is a director who supervises several managers. Each manager supervises several staff members. Most of the jobs can be handled by the staff members, but there are certain jobs that can only be handled by the managers or the director. Moreover, there are other jobs that can only be handled by the director. Thus, we can view the director as the root of the tree, the managers are the descendants of the director, and the staff members are the descendants of the managers. Here we assume that the director is more experienced and efficient than the managers, and the managers are more experienced and efficient than the staff members. Given a set of jobs, we would like to finish the jobs as soon as possible.

Epstein and Levin (2011) give a Polynomial Time Approximation Scheme (PTAS) to solve $Q \mid \mathcal{M}_j(inclusive) \mid C_{\max}$. However, the running time of their PTAS

is very high. Li and Lee (2016) present an $O(mn^2 \log m)$ time algorithm for the problem $Q \mid \mathcal{M}_j(inclusive), r_j, p_j = p \mid C_{\max}$; i.e., a set of equal-processing-time jobs with release times. To the best of our knowledge, there is no fast approximation algorithm to solve the general problem. This paper is a first step towards this direction.

Epstein and Levin (2011) also give a PTAS to solve $Q \mid \mathcal{M}_j(tree) \mid C_{\max}$. Again, the running time of their PTAS is very high. Li and Lee (2016) present an $O(mn^2 \log m)$ time algorithm for the problem $Q \mid \mathcal{M}_j(tree), r_j, p_j = p \mid C_{\max}$; i.e., a set of equal-processing-time jobs with release times. To the best of our knowledge, there is no fast approximation algorithm to solve the general problem.

We will set up some conventions and notations used throughout this paper. There are m uniform machines $\mathcal{M} = \{M_1, M_2, ..., M_m\}$, where machine M_i has speed $s_i > 0$. We assume that $s_1 \leq s_2 \leq ... \leq s_m$. (Note that our algorithm may not work if this assumption is not valid.) Furthermore, we assume that $s_1 = 1$ and $s_i \geq 1$ for all $2 \leq i \leq m$. In the case of "tree-hierarchical processing set", M_m is the root while M_1 is a leaf of the tree.

The organization of this paper is as follows. In the next section, we will review the literature of this and related problems. In Section 3, we give the algorithm for "inclusive processing set" and prove that it has a worst-case performance bound of 4/3. In Section 4, we give the algorithm for "tree-hierarchical processing set" and prove that it has a worst-case performance bound of 4/3. Finally, we give some concluding remarks in the last section.

2 Literature Review

The problems studied in this paper are special cases of the unrelated machine scheduling problem (i.e., $R \mid\mid C_{\max}$). For the problem $R \mid\mid C_{\max}$, Lenstra et al. (1990) give a polynomial-time algorithm with a worst-case bound of 2. Moreover, they show that there is no polynomial-time algorithm with a worst-case bound better than 3/2, unless P = NP. Shchepin and Vakhania (2005) improve the bound to $2 - 1/m$.

The problem $Q \mid \mathcal{M}_j(inclusive) \mid C_{\max}$ is a natural generalization of the problem $P \mid \mathcal{M}_j(inclusive) \mid C_{\max}$; see Leung and Li (2008, 2016) and Lee et al. (2010) for a survey of this problem. Ou et al. (2008) give a PTAS for $P \mid \mathcal{M}_j(inclusive) \mid C_{\max}$, and Li and Wang (2010) extend their work to include job release times. Ou et al. (2008) also give a strongly polynomial algorithm with a worst-case performance bound of $\frac{4}{3} + \epsilon$ for any real number $\epsilon > 0$. Their algorithm can in fact be modified to yield a worst-case bound of $\frac{4}{3}$, but the modified algorithm is no longer strongly polynomial. Huo and Leung (2010) give a faster algorithm with the same worst-case bound to solve a more general class. Their algorithm actually works for "tree-hierachical processing sets", of which "inclusive processing sets" is a special case. As we shall see later, our algorithms are adapted from their algorithm. In preemptive scheduling, Huo et al. (2009) give a polynomail-time algorithm for the problem $P \mid \mathcal{M}_j(inclusive), r_j, pmtn \mid C_{\max}$.

Bar-Noy et al. (2001) give an online (over list) algorithm for the problem $P \mid \mathcal{M}_j(inclusive) \mid C_{\max}$, and show that it has a competitive ratio of $e + 1$. Tan and Zhang (2011) improve the competitive ratios to 2.333 and 2.610 for the 4-machine and 5-machine cases, respectively. Lim et al. (2011) further improve the competitive ratios to 2.294 and 2.501 for these two cases, respectively.

Zhang et al. (2009) study online scheduling of $P \mid \mathcal{M}_j(inclusive) \mid C_{\max}$ with two GoS levels; i.e., machines are divided into two classes where one class can process all the jobs and the other class can process only a subset of jobs. They provide an online algorithm with a competitive ratio of $1 + \frac{m^2-m}{m^2-km+k^2} < \frac{7}{3}$, where k is the number of machines that can process all the jobs.

Lee et al. (2011) study online (over time) scheduling for the problem $P2 \mid \mathcal{M}_j(inclusive), r_j, p_j = p \mid C_{\max}$. They give an optimal algorithm with a competitive ratio of $\sqrt{2}$. Xu and Liu (2015) later give an optimal algorithm with a competitve ratio of $\sqrt{2}$ for arbitrary number of machines; i.e., $P \mid \mathcal{M}_j(inclusive), r_j, p_j = p \mid C_{\max}$.

Hou and Kang (2012) consider online scheduling on uniform machines, where there are k machines with speed s that can schedule all the jobs and $m-k$ machines with speed 1 that can only schedule a subset of jobs. They present an online algorithm with the following competitive ratio. Let $s_1 \in (0,1)$ be the real root of the equation $k^2s^3 + k(2m-2k-1)s^2 + (m-k)(m-2k)s - (m-k)^2 = 0$ and s_2 be the positive root of the equation $ks^2 - (2k-1)s - (m-k) = 0$. When $0 < s < 1$, the competitive ratio is

$$\begin{cases} \max\left\{1 + \frac{k-1}{k}, 1 + \frac{ks+m-k-1}{m-k}\right\}, & \text{if } 0 < s \le s_1; \\ 1 + \frac{(m-1)(ks+m-k)}{k^2s^2+k(m-k)s+(m-k)^2}, & \text{if } s_1 < s < 1. \end{cases}$$

When $s \ge 1$, the competitive ratio is

$$\begin{cases} 1 + \frac{k(k-1)s^2+(m-k)(m+k-1)s}{k^2s^2+k(m-k)s+(m-k)^2}, & \text{if } 1 \le s < s_2; \\ 1 + \frac{(k-1)s+m-k}{ks}, & \text{if } s \ge s_2. \end{cases}$$

Liu *et al.* (2009) study online scheduling of the problem $Q2|\mathcal{M}_j(inclusive)|C_{\max}$.

They assume that M_1 has speed 1 and can process any job, while M_2 has speed $s > 0$ and can process only a subset of jobs. They derive a lower bound on the competitive ratio, and propose two online algorithms. Lee *et al.* (2009) point out an error in Liu *et al.* (2009) and propose several online algorithms for the problem. They derive competitive ratios for these algorithms: Let $s_1 = \frac{\sqrt{5}-1}{2}$ and s_2 be the solution of the equation $s^3 - s - 1 = 0$ for $1.3 < s < 1.4$. When $s \in (0, s_1]$, the High Speed Machine First (HSF) algorithm has a competitive ratio of $1 + s$, and it is optimal. When $s \in [s_2, \infty)$, HSF is also optimal and has a competitive ratio of $1 + \frac{1}{s}$. When $s \in (s_1, 1)$, they propose a "Modified ONLINE1" algorithm and show that it has a competitive ratio of $1 + \frac{2s}{s^2+s+1}$. When $s \in (1, s_2)$, they propose a "Modified ONLINE2" algorithm and show that it has a competitive ratio of $1+\frac{s^2+s}{s^2+s+1}$. Both ONLINE1 and ONLINE2 are not optimal; i.e., their competitive ratios do not match the lower bounds.

Tan and Zhang (2010) consider the same problem as in Lee *et al.* (2009). They assume that M_1 has speed $s > 0$ and can process any job, while M_2 has speed 1 and can process only a subset of jobs. For $s < 1$, they propose an optimal algorithm with a competitive ratio of $\min\{1+s, 1+\frac{1+s}{1+s+s^2}\}$. For $s > 1$, they propose another optimal algorithm with a competitive ratio of $\min\{\frac{1+s}{s}, 1 + \frac{2s}{1+s+s^2}\}$.

Semi-online algorithms have also been studied before, where some partial information about the jobs are known to the scheduler. The interested reader is referred to the survey paper by Leung and Li (2016) for more information.

Epstein and Levin (2011) give a PTAS for $P \mid \mathcal{M}_j(tree) \mid C_{\max}$, while Huo and Leung (2010) give a fast approximation algorithm with a worst-case bound of 4/3. Li and Li (2015) give an $O(mn \log n)$ time algorithm for the problem $P \mid \mathcal{M}_j(tree), r_j, p_j = p \mid C_{\max}$, while Li and Lee (2016) give an $O(mn^2 \log m)$ time algorithm for the problem $Q \mid \mathcal{M}_j(tree), r_j, p_j = p \mid C_{\max}$,

Xu and Liu (2015) consider online scheduling (over time) for the three machine problem $P3 \mid \mathcal{M}_j(tree), r_j, p_j = p \mid C_{\max}$. They design an optimal algorithm with a competitive ratio of 3/2.

3 Inclusive processing sets

In this section we give a fast approximation algorithm for $Q \mid \mathcal{M}_j(inclusive) \mid C_{\max}$. First, we need to set up notations just for this section. There are m uniform machines $\mathcal{M} = \{M_1, M_2, ..., M_m\}$, where machine M_i has speed $s_i > 0$. We assume that $s_1 \leq s_2 \leq ... \leq s_m$. Let there be z classes of speeds $S_1 < S_2 < ... < S_z$, and for each class i there are m_i machines with speed S_i. Without loss of generality, we may assume that $S_1 = 1$ and $S_i > 1$ for all $2 \leq i \leq z$. Thus, $\mathcal{M}$ can be partitioned into z subsets $\mathcal{MS}_1$, $\mathcal{MS}_2$, ..., $\mathcal{MS}_z$, where the machines in $\mathcal{MS}_i$ all have speed S_i, $1 \leq i \leq z$. Let $a[i]$ and $b[i]$ be the indexes of the starting and ending machines in $\mathcal{MS}_i$, $1 \leq i \leq z$. That is, $M_{a[i]}$ and $M_{b[i]}$ are the first and last machines in $\mathcal{MS}_i$. Clearly, $m_i = b[i] - a[i] + 1$ for each $1 \leq i \leq z$.

A set of n jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ is to be scheduled on the m machines, where job J_j has a processing requirement p_j. $\mathcal{J}$ can be partitioned into z subsets $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_z$, where $\mathcal{J}_1$ consists of all jobs that can be processed by any machine in $\mathcal{M}$, $\mathcal{J}_2$ consists of all jobs that can be processed by any machine in $\mathcal{M} \setminus \mathcal{MS}_1$, $\mathcal{J}_3$ consists of all jobs that can be processed by any machine in $\mathcal{M} \setminus \{\mathcal{MS}_1 \cup \mathcal{MS}_2\}$, and so on. Note that some of the $\mathcal{J}_i$ could be the empty set. Without loss of generality, we may assume that $\mathcal{J}_1 \neq \emptyset$; otherwise, we can delete all machines in $\mathcal{MS}_1$ from consideration since no job can be scheduled on those machines.

Our algorithm is actually an adaptation of the algorithm by Huo and Leung (2010), which solves the "tree-hierarchical processing sets" case. Since "inclusive process-

ing sets" is a special case of "tree-hierarchical processing sets" case, their algorithm solves $P \mid \mathcal{M}_j(inclusive) \mid C_{\max}$ as well. Before we give our algorithm, we will first review the algorithm of Huo and Leung (2010).

We use the same notation as in the previous paragraphs. That is, $\mathcal{M}$ is partitioned into z subsets $\mathcal{MS}_1$, $\mathcal{MS}_2$, ..., $\mathcal{MS}_z$, and $\mathcal{J}$ is partitioned into z subsets $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_z$. For the problem $P \mid \mathcal{M}_j(inclusive) \mid C_{\max}$, we may assume that $S_1 = S_2 = ... = S_z$ and $\mathcal{J}_j \neq \emptyset$ for each $1 \leq j \leq z$.

The algorithm of Huo and Leung (2010) first computes an upper bound UB and a lower bound LB for the optimal makespan. The upper bound can be obtained by running the algorithm given by Bar-Noy et al. (2001), and the makespan of such a schedule can be used as UB. The lower bound LB can be set as one half of UB. The algorithm then conducts a binary search in the interval [LB, UB]. For each value C obtained in the binary search, the algorithm tries to schedule the jobs on the machines (by a subroutine) so that the makespan is not more than $4C/3$. If it is successful, it searches the lower half of the range; otherwise, it searches the upper half.

The subroutine that schedules the jobs takes as input an integer parameter C. It returns "Yes" if it is possible to schedule all jobs so that the makespan is $4C/3$ or less; otherwise, it returns "No". For a given value of C, we call job J_j a Type-1 job if its processing requirement p_j is more than $2C/3$. We call J_j a Type-2 job if its processing requirement p_j is more than $C/3$ but not more than $2C/3$. Finally, we call J_j a Type-3 job if its processing requirement p_j is not more than $C/3$. The subroutine schedules the jobs in $\mathcal{J}_k$ onto the machines in $\mathcal{M}_k$ from $k = 1$ until $k = z$. For each k, the subroutine assigns jobs in $\mathcal{J}_k$ to the machines in $\mathcal{M}_k$ in the order from $M_{a[k]}$ to $M_{b[k]}$ as follows.

Suppose we are considering machine M_j, $a[k] \leq j \leq b[k]$. If $\mathcal{J}_k$ has one or more Type-1 jobs, then schedule the longest Type-1 job on M_j. Otherwise, if $\mathcal{J}_k$ has two or more Type-2 jobs, then schedule the two longest Type-2 jobs on M_j. If $\mathcal{J}_k$ has only one Type-2 job, then schedule the Type-2 job on M_j. After this, we schedule the Type-3 jobs, in any order, until either there is no more Type-3 job or we encounter a situation where the scheduling of the job will make the makespan longer than $4C/3$, whichever occurs first. The scheduled jobs will be deleted from $\mathcal{J}_k$. If $\mathcal{J}_k$ becomes the empty set, then we repeat the above process to schedule the jobs in $\mathcal{J}_{k+1}$ on the machines in $\mathcal{MS}_{k+1}$. Otherwise, we consider the machine M_{j+1}. If $j+1 = a[k+1]$, then we merge $\mathcal{J}_k$ into $\mathcal{J}_{k+1}$ before scheduling any job.

Our algorithm also computes a lower bound LB and an upper bound UB of the optimal makespan, and conducts a binary search in the interval [LB, UB]. For each value C encountered in the binary search, we try to schedule the jobs $\mathcal{J}$ on the machines in $\mathcal{M}$ so that the makespan is not larger than $4C/3$. If it is successful, then we search the lower half of the range; otherwise, we search the upper half of the range. However, our LB and UB are computed differently than that of Huo and Leung (2010), since the machines have different speeds.

The scheduling of jobs is more complicated. First, there may be a job $J_j \in \mathcal{J}_k$ such that $p_j/S_k > C$. In this case, J_j cannot be scheduled on any machine in $\mathcal{MS}_k$. Therefore, we will merge J_j into $\mathcal{J}_{k+1}$. Second, Type-1, Type-2 and Type-3 jobs are defined with respect to the speeds of the machines in $\mathcal{MS}_k$. That is, J_j is a Type-1 (or Type-2 or Type-3) job in $\mathcal{MS}_k$ if $p_j/S_k > 2C/3$ (or $C/3 < p_j/S_k \leq 2C/3$ or $p_j/S_k \leq C/3$). Thus, a job J_j may be a Type-1 job with respect to the machines in $\mathcal{MS}_k$, but it may be a Type-2 or Type-3 job with respect to the machines in $\mathcal{MS}_{k+1}$ and beyond.

In the next two subsections, we will describe our method to compute LB and UB,

as well as the feasibility subroutine.

3.1 Lower and Upper Bounds

Define $p_{\max}$ to be the largest processing requirement among all the jobs in $\mathcal{J}$; i.e., $p_{\max} = \max\{p_j \mid J_j \in \mathcal{J}\}$. One possible value of LB is $L_0 = p_{\max}/S_z$. Other possible values of LB are given by L_k, $1 \leq k \leq z$, where $L_k = \frac{\sum_{J_j \in \mathcal{J}_k \cup ... \cup \mathcal{J}_z} p_j}{\sum_{j=k}^{z} m_j * S_j}$. That is, L_k is the level obtained by spreading the jobs in $\mathcal{J}_k \cup \mathcal{J}_{k+1} \cup ... \cup \mathcal{J}_z$ on the machines in $\mathcal{MS}_k \cup \mathcal{MS}_{k+1} \cup ... \cup \mathcal{MS}_z$. Thus, if we take the maximum of L_0, L_1, ..., L_z, we get a lower bound LB. An upper bound can be obtained by spreading all the jobs on the machines in $\mathcal{MS}_z$; i.e., $UB = \frac{\sum_{j=1}^{n} p_j}{m_z * S_z}$.

3.2 Feasibility Test

In this subsection, we describe the procedure to test if a set of jobs can be scheduled on the machines so that the makespan is not larger than $4C/3$ for a given input parameter C. The procedure schedules the jobs in the order of $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_z$, and the machines will be scheduled in the order $M_{a[k]}$, $M_{a[k]+1}$, ..., $M_{b[k]}$. Suppose we are considering the jobs in $\mathcal{J}_k$ and the machine M_i, $a[k] \leq i \leq b[k]$. First, if there is a job J_j such that $p_j/S_k > C$, then it merges J_j into the next set $\mathcal{J}_{k+1}$. Second, it classifies the jobs into Type-1, Type-2 and Type-3 jobs. Job J_j is a Type-1 (respectively Type-2 and Type-3) job if $p_j/S_k > 2C/3$ (respectively $C/3 < p_j/S_k \leq 2C/3$ and $p_j/S_k \leq C/3$). Third, it schedules the jobs as follows. If there is a Type-1 job, then schedule the longest Type-1 job on machine M_i and schedule the Type-3 jobs, in any order, until it either exhausts all Type-3 jobs or it encounters a Type-3 job such that the makespan exceeds $4C/3$, whichever occurs first. If there is no Type-1 job but there are two or more Type-2 jobs, then schedule the two longest

Type-2 jobs and fill in with Type-3 jobs like before. On the other hand, if there is only one Type-2 job, then schedule the (only) Type-2 job and fill in with Type-3 jobs like before. Finally, if there is no Type-1 or Type-2 jobs, then schedule the Type-3 jobs, in any order, until it either exhausts all Type-3 jobs or it encounters a Type-3 job such that the makespan exceeds $4C/3$, whichever occurs first. Afterwards, if $\mathcal{J}_k$ becomes the empty set, then we consider $\mathcal{J}_{k+1}$ and the machines in $\mathcal{MS}_{k+1}$; otherwise, we consider $\mathcal{J}_k$ and the machine M_{i+1}. If $i+1 = a[k+1]$, then merge $\mathcal{J}_k$ into $\mathcal{J}_{k+1}$ before scheduling any job. The feasibility procedure is given as follows.

Feasibility(C)

Input: An integer parameter C, z sets of jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_z$, and z sets of machines $\mathcal{MS}_1$, $\mathcal{MS}_2$, ..., $\mathcal{MS}_z$.

Output: "Yes" if all jobs can be scheduled so that the makespan is not larger than $4C/3$; otherwise, "No".

(1) For k = 1 to z do

  (a) While there is a job $J_j \in \mathcal{J}_k$ such that $p_j/S_k > C$ do $\{\mathcal{J}_{k+1} := \mathcal{J}_{k+1} \cup \{J_j\}$ and $\mathcal{J}_k := \mathcal{J}_k \setminus \{J_j\}\}$.

  (b) Classify the jobs in $\mathcal{J}_k$ as Type-1, Type-2, or Type-3 jobs.

  (c) For i = $a[k]$ to $b[k]$ do

    (i) If there is a Type-1 job in $\mathcal{J}_k$ then { schedule the longest Type-1 job on M_i and remove the job from $\mathcal{J}_k$. }

    Else if there are two Type-2 jobs in $\mathcal{J}_k$ then { schedule the two longest Type-2 jobs on M_i and remove the jobs from $\mathcal{J}_k$. }

    Else if there is one Type-2 job in $\mathcal{J}_k$ then { schedule the Type-2 job on M_i and remove the job from $\mathcal{J}_k$. }

    (ii) While there is a Type-3 job that can be scheduled on M_i without

making the makespan larger than $4C/3$ do { schedule the Type-3 job on M_i and remove the job from $\mathcal{J}_k$. }

(iii) If $\mathcal{J}_k = \emptyset$ then **break.**

(d) If $k < z$ then { $\mathcal{J}_{k+1} := \mathcal{J}_{k+1} \cup \mathcal{J}_k$. }

(2) If $\mathcal{J}_z = \emptyset$ then return "Yes" else return "No". □

Before we leave this subsection, we will give an instance showing that the $4/3$ bound is achievable. This instance has four machines M_1, M_2, M_3 and M_4, with $\mathcal{MS}_1 = \{M_1, M_2\}$ and $\mathcal{MS}_2 = \{M_3, M_4\}$. The speed of the machines in $\mathcal{MS}_1$ is $S_1 = 1$ and the speed of the machines in $\mathcal{MS}_2$ is $S_2 = 1 + \epsilon$, where $\epsilon > 0$ is a small real number. There are eight jobs with $\mathcal{J}_1 = \{J_1, J_2, J_3, J_4\}$ and $\mathcal{J}_2 = \{J_5, J_6, J_7, J_8\}$. For a given integer C, the processing requirements are: $p_1 = p_2 = p_5 = p_6 = 2C/3$ and $p_3 = p_4 = p_7 = p_8 = C/3$. Thus, jobs J_1 and J_2 are Type-2 jobs with respect to the machines in $\mathcal{MS}_1$, and jobs J_5 and J_6 are Type-2 jobs with respect to the machines in $\mathcal{MS}_2$. Similarly, jobs J_3, J_4, J_7 and J_8 are Type-3 jobs.

For the input parameter C, the Feasibility routine will return "Yes" since J_1 and J_2 will be scheduled on M_1, J_3 and J_4 will be scheduled on M_2, J_5 and J_6 will be scheduled on M_3, and J_7 and J_8 will be scheduled on M_4. The makespan of this schedule is $4C/3$. The optimal schedule will have J_1 and J_3 on M_1, J_2 and J_4 on M_2, J_5 and J_7 on M_3, and J_6 and J_8 on M_4. The makespan of the optimal schedule is C. Therefore, the ratio is 4/3.

If we call the Feasibility routine with the input parameter $C' < C$, then jobs J_1 and J_2 become Type-1 jobs while jobs J_3 and J_4 become Type-2 jobs. There is no Type-3 jobs. So, the Feasibility routine will schedule J_1 on M_1 and J_2 on M_2, while the jobs J_3 and J_4 will be pushed to $\mathcal{J}_2$. Assuming ϵ is very small, both J_5 and J_6 will be Type-1 jobs with respect to the machines in $\mathcal{MS}_2$, while J_7 and J_8 are Type-2 jobs with respect to the machines in $\mathcal{MS}_2$. Therefore, J_5 and J_6 will be scheduled

on M_3 and M_4, respectively. Thus, jobs J_3, J_4, J_7 and J_8 are not scheduled and hence the Feasibility routine will return "No". Thus, our algorithm will stop at C since for any $C' < C$, the feasibility routine returns "No".

3.3 Performance Analysis

In this subsection, we will prove that the worst-case performance ratio of our algorithm is 4/3. The example in the last subsection shows that the bound is achievable.

We assert that if the jobs in $\mathcal{J}$ can be scheduled optimally on the machines in $\mathcal{M}$ so that the makespan is C or less, then our algorithm must be able to schedule them with a makespan of $4C/3$ or less. This assertion will be proved by contradiction. Assume that $\mathcal{J}$ and $\mathcal{M}$ form the smallest counterexample; i.e., the jobs in $\mathcal{J}$ can be scheduled on the machines in $\mathcal{M}$ so that the optimal makespan is C or less, but our algorithm fails to schedule them so that the makespan is $4C/3$ or less. Let J_x be the largest unscheduled job in $\mathcal{J}_z$ at the end of the algorithm. By our assumption that $\mathcal{J}$ is the smallest counterexample, we may assume that J_x is the only unscheduled job in $\mathcal{J}_z$ at the end of the algorithm.

There are three cases to consider: J_x is a Type-1, or Type-2, or Type-3 job. In the next three lemmas, we will derive a contradiction in each of these three cases.

Lemma 1: If J_x is a Type-1 job, then the optimal makespan of the jobs in $\mathcal{J}$ on the machines in $\mathcal{M}$ is more than C.

Proof: Since J_x is an unscheduled Type-1 job, it is clear that every machine in $\mathcal{MS}_z$ has one Type-1 job scheduled on it. If all these Type-1 jobs belong to $\mathcal{J}_z$ or they are passed to $\mathcal{J}_z$ from $\mathcal{J}_{z-1}$ because of their size (i.e., their processing requirement divided by S_{z-1} is larger than C), then all these Type-1 jobs must be processed by

the machines in $\mathcal{MS}_z$. Since there are $m_z + 1$ Type-1 jobs and there are only m_z machines in $\mathcal{M}_z$, one of the machines must have two Type-1 jobs scheduled on it in the optimal schedule, implying that the makespan is more than C.

On the other hand, if there is a Type-1 job J_y passed from $\mathcal{J}_{z-1}$ not because of its size, but because it did not get a chance to be scheduled, then since $S_{z-1} < S_z$, J_y must also be a Type-1 job in $\mathcal{J}_{z-1}$. Again, every machine in $\mathcal{MS}_{z-1}$ has one Type-1 job assigned. Since there are $m_{z-1} + 1$ Type-1 jobs but only m_{z-1} machines, the optimal schedule must assign two Type-1 jobs on one machine, implying that the makespan is more than C.

Similarly, if there is a Type-1 job J_w passed from $\mathcal{J}_{z-2}$ to $\mathcal{J}_{z-1}$ not because of its size, but because it did not get a chance to be scheduled, then we can repeat the same argument to show that the optimal makespan must be larger than C. □

Lemma 2: If J_x is a Type-2 job, then the optimal makespan of the jobs in $\mathcal{J}$ on the machines in $\mathcal{M}$ is more than C.

Proof: Since J_x is an unscheduled Type-2 job, every machine in $\mathcal{MS}_z$ has either one Type-1 job or two Type-2 jobs assigned. If all these jobs belong to $\mathcal{J}_z$ or they are passed from $\mathcal{J}_{z-1}$ because of their size, then they must be processed by a machine in $\mathcal{MS}_z$. In order to schedule all the jobs in $\mathcal{J}_z$, the optimal schedule must either have one machine with one Type-1 and one Type-2 jobs, or one machine with three Type-2 jobs. In either case, the optimal schedule must have a makespan larger than C.

If there is a Type-2 job J_y passed from $\mathcal{J}_{z-1}$ not because of its size, but because it did not get a chance to be scheduled, then since $S_{z-1} < S_z$, J_y must be either a Type-1 or Type-2 job in $\mathcal{J}_{z-1}$. In the former case, every machine in $\mathcal{MS}_{z-1}$ has one

Type-1 job assigned, and in the latter case, every machine has either one Type-1 job or two Type-2 jobs assigned. In both cases, the optimal schedule must either assign one Type-1 and one Type-2 jobs, or three Type-2 jobs on one machine. In either case, the schedule must have a makespan larger than C.

Similarly, if there is a Type-2 job J_w passed from $\mathcal{J}_{z-2}$ to $\mathcal{J}_{z-1}$ not because of its size, but because it did not get a chance to be scheduled, then we can repeat the same argument to show that the optimal schedule must have a makespan larger than C. $\square$

Lemma 3: If J_x is a Type-3 job, then the optimal makespan of the jobs in $\mathcal{J}$ on the machines in $\mathcal{M}$ is more than C.

Proof: Since J_x is an unscheduled Type-3 job, every machine in $\mathcal{MS}_z$ has a makespan larger than C; i.e., it is filled to a level more than C. Let $\mathcal{J}'$ denote all the jobs assigned on a machine in $\mathcal{MS}_z$ plus the job J_x. If all the jobs in $\mathcal{J}'$ belong to $\mathcal{J}_z$ or they are passed from $\mathcal{J}_{z-1}$ because of their size, then they must be processed by a machine in $\mathcal{MS}_z$ in the optimal schedule. Since every machine in $\mathcal{MS}_z$ has a makespan larger than C, it is not possible to schedule all the jobs in $\mathcal{J}'$ so that the makespan is C or less.

On the other hand, if there is a job J_y passed from $\mathcal{J}_{z-1}$ not because of its size, but because it did not get a chance to be scheduled, then since $S_{z-1} < S_z$, there are three possibilities for J_y: (A) J_y is a Type-1 job with respect to the machines in $\mathcal{MS}_{z-1}$, (B) J_y is a Type-2 job with respect to the machines in $\mathcal{MS}_{z-1}$, and (C) J_y is a Type-3 job with respect to the machines in $\mathcal{MS}_{z-1}$. For Case (A), every machine in $\mathcal{MS}_{z-1}$ has one Type-1 job assigned. For Case (B), every machine in $\mathcal{MS}_{z-1}$ has one Type-1 or two Type-2 jobs assigned. For Case (C), every machine in $\mathcal{MS}_{z-1}$ has a makespan larger than C; i.e., filled to a level more than C. In

all three cases, we can use the same argument as before to show that the optimal makespan must be larger than C.

Similarly, if there is a Type-3 job J_w passed from $\mathcal{J}_{z-2}$ to $\mathcal{J}_{z-1}$ not because of its size, but because it did not get a chance to be scheduled, then we can use the same argument to show that the optimal schedule must have a makespan larger than C. $\square$

By Lemmas 1, 2 and 3, we have the following theorem.

Theorem 1: There is a fast approximation algorithm for $Q|\mathcal{M}_j(inclusive) \mid C_{\max}$ with a worst-case bound of 4/3.

We now consider the running time of the algorithm. It takes $O(n)$ time to compute the lower and upper bounds. The Feasibility routine can be implemented to run in $O(mn)$ time. The routine will be called at most $\log(UB - LB) \leq \log UB$ times. The largest possible value of UB is $UB \leq \frac{\sum_{j=1}^{n} p_j}{m_z * S_z} \leq \sum_{j=1}^{n} p_j$. Thus, the running time of repeatedly calling the Feasibility routine is $O(\log P * m * n)$, where $P = \sum_{j=1}^{n} p_j$. Including the LB and UB computations, the overal running time of the algorithm is $O(\log P * m * n)$. Thus, we have the following theorem.

Theorem 2: The running time of the algorithm is at most $O(\log P * m * n)$.

4 Tree-hierarchical processing sets

In this section we will give a fast approximation algorithm for $Q \mid \mathcal{M}_j(tree) \mid C_{\max}$. We first define some notations that will be used throughout this section. Let $\mathcal{M} = \{M_1, M_2, ..., M_m\}$ be such that $s_1 \leq s_2 \leq ... \leq s_m$. Without loss of generality, we may assume that $s_1 = 1$ and $s_k \geq 1$ for each $2 \leq k \leq m$. The root of

the tree is M_m and M_1 is a leaf of the tree. For each $1 \leq k \leq m$, let $\mathcal{J}_k$ denote the set of jobs associated with the machine M_k. Note that some of the $\mathcal{J}_k$ may be the empty set. For each machine M_k, we define the level of M_k, denoted by l_k, as follows: If M_k is the root of the tree, then $l_k = 1$; otherwise, $l_k = 1 + l_j$ where l_j is the level of its immediate successor. The height of the tree is the maximum level among all of its vertices, and is denoted by h.

The algorithm first computes an upper bound UB and a lower bound LB for the optimal makespan. An obvious upper bound is $UB = \frac{\sum_{j=1}^{n} p_j}{s_m} \leq P$, where $P = \sum_{j=1}^{n} p_j$, and an obvious lower bound is $LB = 1$. The algorithm then conducts a binary search in the interval [LB, UB]. For each value C obtained in the binary search, it will call the TreeFeasibility routine to determine if we can schedule all the jobs so that the makespan is not larger than $4C/3$. If we can, then we search the lower half of the range; otherwise, we search the upper half.

A job J_j in $\mathcal{J}_k$ is classified as Type-1 (respectively Type-2 and Type-3) job if $p_j/s_k > 2C/3$ (respectively $C/3 < p_j/s_k \leq 2C/3$ and $p_j/s_k \leq C/3$). The TreeFeasibility routine schedules jobs associated with a machine from the highest level to the lowest level. Suppose we are considering the machine M_k. If $\mathcal{J}_k$ has one or more Type-1 jobs, then schedule the longest Type-1 job on M_k. Otherwise, if $\mathcal{J}_k$ has two or more Type-2 jobs, then schedule the two longest Type-2 jobs on M_k. Otherwise, if $\mathcal{J}_k$ has only one Type-2 job, then schedule the (only) Type-2 job on M_k. After this is done, schedule the Type-3 jobs, in any order, until it either exhausts all Type-3 jobs or it encounters a Type-3 job that will exceed $4C/3$, whichever occurs first. We then merge the remaining jobs to $\mathcal{J}_i$, where M_i is the immediate successor of M_k. When we reach the root, if all jobs are scheduled, then we return "Yes"; otherwise, we return "No". The TreeFeasibility routine is described as follows.

TreeFeasibility(C)

Input: A rooted tree with vertices $\mathcal{M} = \{M_1, M_2, ..., M_m\}$. Each vertex M_k has a set of jobs $\mathcal{J}_k$ assoicated with it. The height of the tree is h.

Output: "Yes" if all jobs can be scheduled so that the makespan is $4C/3$ or less; otherwise, "No".

(1) While ($h > 1$) do

  (a) For each machine M_k with $l_k = h$ do

    (i) If there is a Type-1 job in $\mathcal{J}_k$ then schedule the longest Type-1 job on M_k and remove it from $\mathcal{J}_k$.

    Else if there are two Type-2 jobs in $\mathcal{J}_k$ then schedule the two longest Type-2 jobs on M_k and remove both of them from $\mathcal{J}_k$.

    Else if there is only one Type-2 job in $\mathcal{J}_k$ then schedule the (only) Type-2 job on M_k and remove it from $\mathcal{J}_k$.

    (ii) While the next Type-3 job J_j can be scheduled on M_k so that the makespan is not larger than $4C/3$ do { Schedule J_j on M_k and remove J_j from $\mathcal{M}_k$.

    (iii) Merge $\mathcal{J}_k$ into $\mathcal{J}_i$, where M_i is the immediate successor of M_k.

  (b) $h := h - 1$.

(2) If $\sum_{j \in \mathcal{J}_m} p_j \leq 4C/3$ then return "Yes"; otherwise, return "No". □

Before we prove that the worst-case performance bound of our algorithm is 4/3, we will give an instance showing that the 4/3 bound is achievable. This instance has four machines, M_1 to M_4. M_1 and M_2 are the leaves of the tree. They are both predecessors of M_3 which in turn is a predecessor of M_4. M_4 is the root of the tree. The speeds of M_1 and M_2 are 1, while the speeds of M_3 and M_4 are $1 + \epsilon$, where $\epsilon > 0$ is a small real number. There are eight jobs J_1 to J_8. The four jobs J_1 to J_4

are associated with M_1; i.e., $\mathcal{J}_1 = \{J_1, J_2, J_3, J_4\}$. Similarly, $\mathcal{J}_2 = \{J_5, J_6, J_7, J_8\}$, and $\mathcal{J}_3 = \mathcal{J}_4 = \emptyset$. For a given integer C, the processing requirements of J_1, J_2, J_5 and J_6 are $2C/3$, while that of J_3, J_4, J_7 and J_8 are $C/3$. Thus, J_1 and J_2 are regarded as Type-2 jobs with respect to M_1, while J_3 and J_4 are regarded as Type-3 jobs. Similarly, J_5 and J_6 are regarded as Type-2 jobs with respect to M_2, while J_7 and J_8 are regarded as Type-3 jobs.

Our algorithm will schedule J_1 and J_2 on M_1, and pass J_3 and J_4 to $\mathcal{J}_3$. Similarly, it will schedule J_5 and J_6 on M_2 and pass J_7 and J_8 to $\mathcal{J}_3$. When the algorithm considers M_3, it will schedule all four jobs J_3, J_4, J_7 and J_8 on M_3, leaving nothing to schedule on M_4. Thus, the algorithm will return "Yes", and the makespan is $4C/3$. The optimal schedule will schedule J_1 and J_3 on M_1, J_5 and J_7 on M_2, J_2 and J_4 on M_3, and J_6 and J_8 on M_4. The makespan of the optimal schedule is C. Thus, the ratio is 4/3.

For any integer $C' < C$, J_1 and J_2 will be regarded as Type-1 jobs with respect to M_1, while J_3 and J_4 will be regarded as Type-2 jobs. Similarly, J_5 and J_6 will be regarded as Type-1 jobs with respect to M_2, while J_7 and J_8 will be regarded as Type-2 jobs. Thus, our algorithm will schedule J_1 on M_1, and pass J_2, J_3 and J_4 to M_3. Similarly, it will schedule J_5 on M_2, and pass J_6, J_7 and J_8 to M_3. When we consider M_3, the algorithm will schedule J_2 on M_3, and pass J_3, J_4, J_6, J_7 and J_8 to M_4. The five jobs in $\mathcal{J}_4$ now have total processing requirements more than $4C/3$. Hence, our algorithm returns "No". In other words, our algorithm will return "No" for any integer $C' < C$. Hence our algorithm will stop at C.

We now turn our attention to proving the worst-case performance bound of our algorithm. We will show that if the optimal schedule can schedule all the jobs with a makespan not more than C, then our algorithm will also be able to schedule the jobs with a makespan not more than $4C/3$. We prove this fact by contradiction. Assume

$\mathcal{J}$ and $\mathcal{M}$ form the smallest counterexample that violates this claim. We assume that the scheduling of jobs on the root vertex is done in exactly the same manner as any other vertices. Let J_x be the largest unscheduled job. By our assumption that $\mathcal{J}$ and $\mathcal{M}$ form the smallest counterexample, we may assume that J_x is the only unscheduled job in $\mathcal{J}_m$.

We will consider three cases: J_x is a Type-1, or Type-2, or Type-3 job. We will show a contradiction in each of these three cases in the next three lemmas.

Lemma 4: If J_x is a Type-1 job, then the optimal makespan is more than C.

Proof: Since J_x is an unscheduled Type-1 job, it is clear that M_m has one Type-1 job scheduled on it. If all these Type-1 jobs belong to $\mathcal{J}_m$ or they are passed to M_m from its predecessor M_l because of their size (i.e., their processing requirement divided by s_l is larger than C), then all these Type-1 jobs must be processed by M_m. Since there are two Type-1 jobs in $\mathcal{J}_m$ and there is only one machine, M_m must have two Type-1 jobs scheduled on it in the optimal schedule, implying that the makespan is more than C.

On the other hand, if there is a Type-1 job J_y passed from M_l, a predecessor of M_m, not because of its size, but because it did not get a chance to be scheduled, then since $s_l \leq s_m$, J_y must also be a Type-1 job in $\mathcal{J}_l$. Again, M_l must also have one Type-1 job assigned. This means that the optimal schedule must assign two Type-1 jobs on one machine, implying that the makespan is more than C.

Similarly, if there is a Type-1 job J_w passed from $\mathcal{J}_k$ to $\mathcal{J}_l$ not because of its size, but because it did not get a chance to be scheduled, then we can repeat the same argument to show that the optimal makespan must be larger than C. $\square$

Lemma 5: If J_x is a Type-2 job, then the optimal makespan is more than C.

Proof: Since J_x is an unscheduled Type-2 job, M_m has either one Type-1 job or two Type-2 jobs assigned. If all these jobs belong to M_m or they are passed from its predecessor because of their size, then they must be processed by M_m. In order to schedule all the jobs in $\mathcal{J}_m$, the optimal schedule must either have one Type-1 and one Type-2 jobs, or three Type-2 jobs. In either case, the optimal schedule must have a makespan larger than C.

If there is a Type-2 job J_y passed from M_l, a predecessor of M_m, not because of its size, but because it did not get a chance to be scheduled, then since $s_l < s_m$, J_y must be either a Type-1 or Type-2 job in $\mathcal{J}_l$. In the former case, M_l has one Type-1 job assigned, and in the latter case, M_l has either one Type-1 job or two Type-2 jobs assigned. In both cases, the optimal schedule must either assign two Type-1 jobs, or one Type-1 and one Type-2 jobs, or three Type-2 jobs on M_l, implying that the schedule has a makespan larger than C.

Similarly, if there is a Type-2 job J_w passed from M_k, a predecessor of M_l, not because of its size, but because it did not get a chance to be scheduled, then we can repeat the same argument to show that the optimal schedule must have a makespan larger than C. $\square$

Lemma 6: If J_x is a Type-3 job, then the optimal makespan is more than C.

Proof: Since J_x is an unscheduled Type-3 job, M_m has a makespan larger than C; i.e., it is filled to a level more than C. Let $\mathcal{J}'$ denote all the jobs assigned on M_m plus the job J_x. If all the jobs in $\mathcal{J}'$ belong to M_m or they are passed from its predecessor because of their size, then they must be processed by M_m in the optimal schedule. Since M_m has a makespan larger than C, it is not possible to

schedule all the jobs in $\mathcal{J}'$ so that the makespan is C or less.

On the other hand, if there is a job J_y passed from M_l, a predecessor of M_m, not because of its size, but because it did not get a chance to be scheduled, then since $s_l \leq s_m$, there are three possibilities for J_y: (A) J_y is a Type-1 job with respect to M_l, (B) J_y is a Type-2 job with respect to M_l, and (C) J_y is a Type-3 job with respect to M_l. For Case (A), M_l has one Type-1 job assigned. For Case (B), M_l has one Type-1 or two Type-2 jobs assigned. For Case (C), M_l has a makespan larger than C; i.e., filled to a level more than C. In all three cases, we can use the same argument as before to show that the optimal makespan must be larger than C.

Similarly, if there is a Type-3 job J_w passed from M_k, a predecessor of M_l, not because of its size, but because it did not get a chance to be scheduled, then we can use the same argument to show that the optimal schedule must have a makespan larger than C. $\square$

By Lemmas 4, 5 and 6, we have the following theorem.

Theorem 3: There is a fast approximation algorithm for $Q|\mathcal{M}_j(tree) \mid C_{\max}$ with a worst-case bound of 4/3.

We now consider the running time of our algorithm. The TreeFeasibility routine is called $O(\log P)$ times, where $P = \sum_{j=1}^{n} p_j$. Each call takes $O(mn)$ time. Thus, the overall running time of our algorithm is $O(\log P * m * n)$. Hence we have the following theorem.

Theorem 4: The running time of the algorithm is $O(\log P * m * n)$.

5 Conclusion

In this paper we have given fast approximation algorithms for the "inclusive processing sets" and "tree-hierarchical processing sets" cases. Both algorithms have running time $O(\log P * m * n)$ and a worst-case bound of 4/3. Moreover, both bounds are achievable.

Our algorithm is not qualified to be called *strongly polynomial* since the running time is a function of $\log P$. However, we can use the technique developed by Ou et al. (2008) to get a strongly polynomial algorithm. The worst-case bound then becomes $\frac{4}{3} + \epsilon$, where $\epsilon > 0$ is any real number. The idea of the technique of Ou et al. (2008) is to divide the interval [LB, UB] into a number of smaller subintervals, and to conduct a binary search on the smaller subintervals.

For uniform machines, the optimal makespan can be a real number; i.e., not an integer. However, our algorithm produces a schedule based on an integral makespan. Thus, our algorithm may produce a solution that is $\frac{4}{3}\lceil C^* \rceil$, where C^* is the optimal makespan.

For the problem $Q \mid \mathcal{M}_j(inclusive) \mid C_{\max}$, we have made the assumption that $s_1 \leq s_2 \leq ... \leq s_m$. This is because of our application, where a job must be completed within a certain amount of time once it starts its processing. That is, each job J_j has a minimum speed level l_j and it can only be run on machine M_k if $s_k \geq l_j$. For this application, we can sort the machines in increasing order of their speeds and a job J_j will be associated with the first machine M_k such that $s_k \geq l_j$. For the problem $Q \mid \mathcal{M}_j(tree) \mid C_{\max}$, we have made the assumption that the speed of a node is at least that of its predecessor. Again, this is because of our application. Recall the hierarchical management structure where the top level person is the most

experienced person and the low level person is the least experienced one. It thus makes sense to assume that the top level person has the highest speed while the low level person has the lowest speed. If these speed assumptions are not valid, our algorithm may not work. It will be interesting to see if there are other approximation algorithms with a reasonable worst-case bound.

For future research, it will be interesting to see if we can develop a fast approximation algorithm with a worst-case bound better than 4/3. Another interesting problem is to develop a fast approximation algorithm for $Q \mid \mathcal{M}_j(nested) \mid C_{\max}$.

Acknowledgement

The authors would thank the two anonymous referees whose suggestions have greately improved the readability of the paper. The work of the second author is supported in part by the University Grants Council under grant number PolyU 152148/15E.

References

Bar-Noy, A., Freund, A., Naor, J., 2001. Online load balancing in a hierarchical server topology. SIAM Journal on Computing 31 (2), 527–549.

Epstein, L., Levin, A., 2011. Scheduling with processing set restrictions: PTAS results for several variants. International Journal of Production Economics 133 (2), 586–595.

Hou, L.-Y., Kang, L., 2012. Online scheduling on uniform machines with two hierarchies. Journal of Combinatorial Optimization 24 (4), 593–612.

Huo, Y., Leung, J.Y.-T., 2010. Fast approximation algorithms for job scheduling with processing set restrictions. Theoretical Computer Science 411 (44–46), 3947–3955.

Huo, Y., Leung, J.Y-T., Wang, X., 2009. A fast preemptive scheduling algorithm with release times and inclusive processing set restrictions. Discrete Optimization 6 (3), 292-298.

Lee, K., Leung, J.Y.-T., Pinedo, M.L., 2009. Online scheduling on two uniform machines subject to eligibility constraints. Theoretical Computer Science 410 (38–40), 3975–3981.

Lee, K., Leung, J.Y.-T., Pinedo, M.L., 2010. Makespan minimization in online scheduling with machine eligibility. 4OR 8 (4), 331–364.

Lee, K., Leung, J.Y.-T., Pinedo, M.L., 2011. Scheduling jobs with equal processing times subject to machine eligibility constraints. Journal of Scheduling 14 (1), 27–38.

Lenstra, J.K., Shmoys, D., Tardos, E., 1990. Approximation algorithms for scheduling unrelated machines. Mathematical Programming 46, 259-271.

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

Leung, J.Y.-T., Li, C.-L., 2016. Scheduling with processing set restrictions: A literature update. International Journal of Production Economics 175, 1–11.

Li, C.-L., Li, Q., 2015. Scheduling jobs with release dates, equal processing times, and inclusive processing set restrictions. Journal of the Operational Research Society 66 (3), 516–523.

Li, C.-L., Lee, K., 2016. A note on scheduling jobs with equal processing times and inclusive processing set restrictions. Journal of Operational Research Society 67 (1), 83-86.

Li, C.-L., Wang, X., 2010. Scheduling parallel machines with inclusive processing set restrictions and job release times. Europen Journal of Operational Research 200 (3), 702-710.

Lim, K., Lee, K., Chang, S.Y., 2011. Improved bounds for online scheduling with

eligibility constraints. Theoretical Computer Science 412 (39), 5211–5224.

Liu, M., Xu, Y., Chu, C., Zheng, F., 2009. Online scheduling on two uniform machines to minimize the makespan. Theoretical Computer Science 410 (21–23), 2099–2109.

Ou, J., Leung, J.Y.-T., Li, C.-L., 2008. Scheduling parallel machines with inclusive processing set retrictions. Naval Research Logistics 55 (4), 328-338.

Shchepin, E.V., Vakhania, N., 2005. An optimal rounding gives a better approximationfor scheduling unrelated machines. Operations Research Letters 33, 127-133.

Tan, Z., Zhang, A., 2010. A note on hierarchical scheduling on two uniform machines. Journal of Combinatorial Optimization 20 (1), 85–95.

Tan, Z., Zhang, A., 2011. Online hierarchical scheduling: An approach using mathematical programming. Theoretical Computer Science 412 (3), 246–256.

Xu, J., Liu, Z., 2015. Online scheduling with equal processing times and machine eligibility constraints. Theoretical Computer Science 572, 58–65.

Zhang, A., Jiang, Y., Tan, Z., 2009. Online parallel machines scheduling with two hierarchies. Theoretical Computer Science 410 (38-40), 3597–3605.