Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/mbe

MBE, 19(9): 9697–9708.
DOI: 10.3934/mbe.2022450
Received: 27 February 2022
Revised: 26 May 2022
Accepted: 13 June 2022
Published: 04 July 2022

Research article

Scheduling uniform machines with restricted assignment

Shuguang Li*and Zhimeng Liu

College of Computer Science and Technology, Shandong Technology and Business University, Yantai 264005, China

* **Correspondence:** Email: sgliytu@hotmail.com; Tel: +8618753509226.

Abstract: The problem of minimizing makespan (maximum completion time) on uniform machines with restricted assignment is considered. The machines differ in their speeds and functionalities. Each job has a set of machines to which it can be assigned, called its processing set. The goal is to finish the jobs as soon as possible. There exist 4/3-approximation algorithms for the cases of inclusive and tree-hierarchical assignment restrictions, under an assumption that machines with higher capabilities also run at higher speeds. We eliminate the assumption and present algorithms with approximation ratios 2 and 4/3 for both cases.

Keywords: scheduling; uniform machines; inclusive restriction; tree-hierarchical restriction; makespan

1. Introduction

Scheduling jobs on (identical, uniform, or unrelated) parallel machines is one of the fundamental problems in deterministic scheduling theory [1–3]. In this paper, the problem of scheduling uniform machines with restricted assignment is studied, which generalizes the problem of scheduling jobs with unrestricted assignment on parallel machines.

Formally, there is a set of jobs $\mathcal{J} = \{1, 2, \dots, n\}$, and there is a set of uniform machines $\mathcal{M} = \{M_1, M_2, \dots, M_m\}$. Job $j \in \mathcal{J}$ has a *length* $p_j \geq 0$ and a subset of machines $\mathcal{M}_j \subseteq \mathcal{M}$ to which it can be assigned, called its *processing set*. Machine $M_i \in \mathcal{M}$ has a *speed* s_i. Without loss of generality, we assume that all $s_i \geq 1$. If job j is processed on machine M_i, then its *processing time* is p_j/s_i. A (non-preemptive) schedule is an m-tuple $(\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_m)$, where $\mathcal{S}_i$ denotes the set of the jobs processed on machine M_i, $i = 1, 2, \dots, m$. The sets $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_m$ are disjoint, and $\bigcup_{i=1}^m \mathcal{S}_i = \mathcal{J}$, i.e., each job in $\mathcal{J}$ appears in exactly one $\mathcal{S}_i$. The *completion time* $C(\mathcal{S}_i)$ of machine M_i in this schedule is given by $\sum_{j\in\mathcal{S}_i} p_j/s_i$. The goal is to find a schedule to minimize the *makespan*, $C_{\max} = \max_i C(\mathcal{S}_i)$. Using the notations proposed in [4, 5], the problem is denoted as $Q|\mathcal{M}_j|C_{\max}$.

As stated in the survey paper by Leung and Li [6], there are five types of assignment restrictions studied by the researchers: *inclusive*, *nested*, *interval*, *tree-hierarchical* and *arbitrary*. In this paper, we focus on the inclusive and tree-hierarchical restrictions, i.e., we study $Q|\mathcal{M}_j(inclusive)|C_{\max}$ and $Q|\mathcal{M}_j(tree)|C_{\max}$. The assignment restriction is *inclusive*, if for any job j, $\mathcal{M}_j = \{M_{a_j}, M_{a_j+1}, \ldots, M_m\}$, where a_j $(1 \le a_j \le m)$ is called the *machine index* associated with job j . The assignment restriction is *tree-hierarchical*, if there is a tree whose nodes represent the machines, and each job j is associated with a tree node M_{a_j}, such that $\mathcal{M}_j$ consists of the machines on the path from M_{a_j} to the root of the tree. Clearly, the inclusive restriction is a special case of the tree-hierarchical restriction.

Leung and Ng [7] studied the problems $Q|\mathcal{M}_j(inclusive)|C_{\max}$ and $Q|\mathcal{M}_j(tree)|C_{\max}$ under a reasonable assumption that machines with higher capabilities also run at higher speeds. (It is called the *speed hierarchical* model in [8]). Precisely, for the case of inclusive restriction, they assumed that $s_1 \le s_2 \le \cdots \le s_m$. For the case of tree-hierarchical restriction, they assumed that the speed of each node is not less than that of its predecessor. Under the assumption, they presented 4/3-approximation algorithms running in time $O(mn \cdot \log(\sum_{j=1}^{n} p_j))$ for both cases. They remarked that the algorithms may not work if the assumption is not valid. For clarity, we denote their problems as $Q(Inc)|\mathcal{M}_j(inclusive)|C_{\max}$ and $Q(Inc)|\mathcal{M}_j(tree)|C_{\max}$, respectively, where “Inc” means “increasing order of the speeds”.

In this paper, we generalize the results in [7] by eliminating the speed hierarchical assumption. We present fast algorithms with approximation ratios 2 and 4/3 for both $Q|\mathcal{M}_j(inclusive)|C_{\max}$ and $Q|\mathcal{M}_j(tree)|C_{\max}$.

Scheduling with restricted assignment has been extensively studied in the literature [6]. Here, we review the results which are related to inclusive or tree-hierarchical restrictions. Ou et al. [9] gave a PTAS (polynomial time approximation scheme) for $P|\mathcal{M}_j(inclusive)|C_{\max}$ (the special case of $Q|\mathcal{M}_j(inclusive)|C_{\max}$ where all $s_i = 1$). Li and Wang [10] extended their work to include job release times (any job cannot be scheduled before its release time). There are also fast approximation algorithms for this problem: a $(2 - 1/(m-1))$ -approximation algorithm [11,12], a 3/2-approximation algorithm [13], and a 4/3-approximation algorithm [9]. Bar-Noy et al. [14] presented an online (over list) algorithm for $P|\mathcal{M}_j(inclusive)|C_{\max}$ whose competitive ratio is $e + 1$. They also gave an online algorithm for $P|\mathcal{M}_j(tree)|C_{\max}$ (the special case of $Q|\mathcal{M}_j(tree)|C_{\max}$ where all $s_i = 1$) whose competitive ratio is 5. Huo and Leung [15] gave a 4/3-approximation algorithm for $P|\mathcal{M}_j(tree)|C_{\max}$. Later, Epstein and Levin [8] presented PTASs for $P|\mathcal{M}_j(tree)|C_{\max}$ and $Q(Inc)|\mathcal{M}_j(inclusive)|C_{\max}$. However, the running times of their PTASs are rather high. Leung and Ng [7] presented fast 4/3-approximation algorithms for $Q(Inc)|\mathcal{M}_j(inclusive)|C_{\max}$ and $Q(Inc)|\mathcal{M}_j(tree)|C_{\max}$. There are also several papers which studied the problems of scheduling jobs with equal lengths and restricted assignment on uniform machines [16–20].

The rest of the paper is organized as follows. In Section 2, we present 2-approximation algorithms for $Q|\mathcal{M}_j(inclusive)|C_{\max}$ and $Q|\mathcal{M}_j(tree)|C_{\max}$. In Section 3, we present 4/3-approximation algorithms for $Q|\mathcal{M}_j(inclusive)|C_{\max}$ and $Q|\mathcal{M}_j(tree)|C_{\max}$. In Section 4, we conclude this paper and discuss future research directions.

2. 2-approximation algorithms

In this section we will get a 2-approximation algorithm for $Q|\mathcal{M}_j(inclusive)|C_{\max}$ and then extend it to solve $Q|\mathcal{M}_j(tree)|C_{\max}$. Note that Leung and Ng [7] only presented algorithms for $Q(Inc)|\mathcal{M}_j(inclusive)|C_{\max}$ and $Q(Inc)|\mathcal{M}_j(tree)|C_{\max}$.

For solving $Q|\mathcal{M}_j(inclusive)|C_{\max}$, we set up some notations. Let $\mathcal{J}_i$ denote the set of the jobs whose processing set is $\{M_i, M_{i+1}, \ldots, M_m\}$, $i = 1, 2, \ldots, m$. The jobs in $\mathcal{J}_i$ are *eligible* for machines $M_i, M_{i+1}, \ldots, M_m$, and vice versa. It is possible that $\mathcal{J}_i = \emptyset$ for some i. We have: $\bigcup_{i=1}^{m} \mathcal{J}_i = \mathcal{J}$. Sort all the jobs in $\mathcal{J}_i$ in non-increasing order of their lengths, and let J_i denote the obtained ordered set, $i = 1, 2, \ldots, m$.

Let OPT denote the makespan of an optimal schedule for $Q|\mathcal{M}_j(inclusive)|C_{\max}$. Let $AL_i = \frac{\sum_{j \in \mathcal{J}_i \cup \mathcal{J}_{i+1} \cup \cdots \cup \mathcal{J}_m} p_j}{\sum_{l=i}^{m} s_l}$ denote the average load on machines $M_i, M_{i+1}, \ldots, M_m$, $i = 1, 2, \ldots, m$. Let $LB = \max_i AL_i$, where "LB" means "lower bound". Since all jobs in $\mathcal{J}_i \cup \mathcal{J}_{i+1} \cup \cdots \cup \mathcal{J}_m$ must be processed on machines $M_i, M_{i+1}, \ldots, M_m$ ($i = 1, 2, \ldots, m$) in any feasible schedule, we get: $LB \le OPT$. On the other hand, let $UB = \frac{\sum_{j=1}^{n} p_j}{s_m}$, where "$UB$" means "upper bound". Since a feasible schedule can be obtained easily by scheduling all the jobs on machine M_m, we get $OPT \le UB$.

To determine OPT, we do a binary search in time interval $[LB, UB]$. For each value C selected, the following procedure, AssignA1, tells us whether it is possible to assign the jobs in $\mathcal{J}$ to the machines in $\mathcal{M}$ such that the total length of the jobs assigned to machine M_i is no more than $2s_i \cdot C$, $i = 1, 2, \ldots, m$. If AssignA1 fails, we search the upper half of the interval; otherwise, we search the lower half.

If job j is eligible for machine M_i and $p_j \le s_i \cdot C$, then job j is *feasible* for machine M_i, and vice versa. In AssignA1, U_i represents the set of unassigned jobs which are eligible for machine M_i and sorted in non-increasing order of their lengths, and L_i denotes the total length of the jobs assigned to machine M_i, $i = 1, 2, \ldots, m$. Informally speaking, AssignA1 tries to put as many as possible of the largest, not yet assigned feasible jobs on the smaller-indexed machines such that each machine completes no later than $2C$.

AssignA1 (C):

```
Step 1. Let $U_0 = \emptyset$, $L_i = 0$, $i = 1, 2, \ldots, m$.
Step 2. For $i = 1, 2, \ldots, m$ (this ordering is used crucially), do:
    (i) Merge $U_{i-1}$ into $J_i$ to get $U_i$. ($U_{i-1}$ and $J_i$ are ordered sets, and thus $U_i$ is also an ordered set.)
    (ii) Find the first job $j \in U_i$ such that $p_j \le s_i \cdot C$. Assign job $j$ and the jobs after it in $U_i$ to
        machine $M_i$ until $L_i > s_i \cdot C$ or the jobs after $j$ in $U_i$ are used up (i.e., each unassigned job in $U_i$
        has length larger than $s_i \cdot C$). Delete the newly assigned jobs from $U_i$.
```

Lemma 2.1. *If $OPT \le C$, then AssignA1 will generate a feasible schedule for $Q|\mathcal{M}_j(inclusive)|C_{\max}$ with makespan at most $2C$ in $O(mn)$ time.*

Proof. Let Π^* be an optimal schedule. Let Π be the schedule generated by AssignA1. Since $OPT \le C$, any job of length larger than $s_i \cdot C$ cannot be assigned on machine M_i in Π^*, $i = 1, 2, \ldots, m$. Therefore, in Π, we let M_i process only the jobs whose lengths are no more than $s_i \cdot C$. We have $L_i \le 2s_i \cdot C$, and thus $C_i \le 2C$, where C_i denotes the completion time of machine M_i in Π, $i = 1, 2, \ldots, m$.

We prove the following claim by contradiction: If $OPT \le C$, then $U_m = \emptyset$ when AssignA1 terminates. Suppose that when AssignA1 terminates, some job j cannot be assigned and has to be left over. Let $a_j = i$ denote the machine index associated with job j . Let $FM_j \subseteq \{M_i, M_{i+1}, \ldots, M_m\}$ denote the set of the feasible machines for job j, i.e., for any machine $M_l \in FM_j$, $p_j \leq s_l \cdot C$. It must be true that $L_l > s_l \cdot C$ for any machine $M_l \in FM_j$. Let D denote the set of the jobs processed on the machines in FM_j. In Π, all the jobs in D have lengths not less than p_j and hence cannot be assigned to the machines in $\{M_i, M_{i+1}, \ldots, M_m\} \backslash FM_j$. We consider the following two different cases:

Case 1. The machine indices associated with the jobs in D are not less than $a_j = i$.

In this case, in any optimal schedule and particularly in Π^*, all the jobs in D have to be processed on the machines in FM_j . However, since $L_l > s_l \cdot C$ for any machine $M_l \in FM_j$, Π^* cannot complete all the jobs in D by time $C \geq OPT$, a contradiction.

Case 2. Some jobs in D have machine indices less than $a_j = i$.

Some jobs in D may be associated with machine indices less than i, but they may be not feasible for machines $M_1, M_2, \ldots, M_{i-1}$, or their feasible machines among $M_1, M_2, \ldots, M_{i-1}$ have not enough space left to accommodate them when they are assigned. Since AssignA1 assigns the largest feasible jobs on the smaller-indexed machines greedily such that each machine completes later than C whenever possible (i.e., unless there is no unassigned feasible job for the machine), an optimal schedule cannot schedule the jobs any better than this on the smaller-indexed machines. Therefore, the total length of the jobs processed on the machines in FM_j in Π is a lower bound on the total length of the jobs processed on the machines in FM_j in Π^*. As in Case 1, since $L_l > s_l \cdot C$ for any machine $M_l \in FM_j$, Π^* cannot complete all the jobs in D by time $C \geq OPT$, a contradiction.

Step 1 can be executed in $O(m)$ time. Step 2 will be executed m iterations, and each iteration can be done in $O(n)$ time. Hence, the running time of AssignA1 is $O(mn)$.

□

To obtain a faster algorithm, we can use the following AssignA2 procedure instead of AssignA1. AssignA2 handles the machines in increasing order of their indices. When M_i is handled, the unassigned jobs eligible for M_i are stored in a balanced binary search tree T [21], using their lengths as the keys. The technique is similar to that used in [20].

AssignA2 (C)**:**

Step 1. Let T be a balanced binary search tree and τ be the length of the smallest job in T. Initially, T is empty, and $\tau = \infty$. Let L_i denote the total length of the jobs assigned to machine M_i. Initially, $L_i = 0, i = 1, 2, \ldots, m$.

Step 2. For $i = 1, 2, \ldots, m$ (this ordering is used crucially), do:

(i). Insert the jobs in $\mathcal{J}_i$ into T. Update τ accordingly.

(ii) If $L_i \leq s_i \cdot C$ and $\tau \leq s_i \cdot C$, then find the largest job j in T such that $p_j \leq s_i \cdot C$. Assign job j on machine M_i and then delete it from T. Let $L_i = L_i + p_j$. If $p_j = \tau$, then check τ and update it if necessary.

(iii) Repeat Step 2(ii) until $L_i > s_i \cdot C$ or $\tau > s_i \cdot C$ (the latter case indicates that each job in T has length larger than $s_i \cdot C$).

Lemma 2.2. *If $OPT \leq C$, then AssignA2 will generate a feasible schedule for $Q|\mathcal{M}_j(inclusive)|C_{\max}$ with makespan at most $2C$ in $O(m + n \log n)$ time.*

Proof. The correctness of AssignA2 follows that of AssignA1, and its proof is omitted. Step 1 can be executed in $O(m)$ time. In Step 2, each job will be inserted into and deleted from T at most once. Inserting or deleting a job can be done in $O(\log n)$ time. Thus, it takes $O(n \log n)$ time to construct and maintain T and τ. In Step 2(ii), it takes $O(\log n)$ time to find the largest job j in T such that $p_j \leq s_i \cdot C$. Hence, the running time of AssignA2 is $O(m + n \log n)$.

□

AssignA1 or AssignA2 will be called at most $\log UB$ times in the binary search. Since $UB = \frac{\sum_{j=1}^{n} p_j}{s_m} \leq \sum_{j=1}^{n} p_j$ (we assumed that all $s_i \geq 1$), we get the following:

Theorem 2.3. *There is a 2-approximation algorithm for $Q|\mathcal{M}_j(inclusive)|C_{\max}$ that runs in $O(min\{mn, m + n \log n\} \cdot \log(\sum_{j=1}^{n} p_j))$ time.*

Next, we extend the algorithm to solve $Q|\mathcal{M}_j(tree)|C_{\max}$. For the rooted tree RT whose nodes are the m machines, we define the depths of the nodes as follows. If the node is the root, then its depth is zero; otherwise, its depth is equal to the depth of its parent plus 1. Index the nodes (machines) of the tree in non-increasing order of their depths, ties broken in favor of the leftmost node. The root of the tree is M_m.

Let $\mathcal{J}_i$ denote the set of the jobs associated with machine M_i, $i = 1, 2, \ldots, m$. The jobs in $\mathcal{J}_i$ are *eligible* for the machines on the path from M_i to the root of the tree, and vice versa. We have: $\bigcup_{i=1}^{m} \mathcal{J}_i = \mathcal{J}$. Sort all the jobs in $\mathcal{J}_i$ in non-increasing order of their lengths, and let J_i denote the obtained ordered set, $i = 1, 2, \ldots, m$.

Let OPT denote the makespan of an optimal schedule for $Q|\mathcal{M}_j(tree)|C_{\max}$. For machine M_i (which represents a node of RT), let I_i denote the set of the indices of the machines on the path from M_i to the root of the tree. Let $AL_i = \frac{\sum_{l \in I_i} \sum_{j \in \mathcal{J}_l} p_j}{\sum_{l \in I_i} s_l}$ denote the average load on the machines whose indices are in I_i, $i = 1, 2, \ldots, m$. Let $LB = \max_i AL_i$. We have: $LB \leq OPT$. Let $UB = \frac{\sum_{j=1}^{n} p_j}{s_m}$. We have: $OPT \leq UB$.

To determine OPT, we do a binary search in interval $[LB, UB]$. For each value C selected, the following procedure, AssignB, tells us whether it is possible to assign the n jobs to the m machines such that the total length of the jobs assigned to machine M_i is no more than $2s_i \cdot C$, $i = 1, 2, \ldots, m$. If AssignB fails, we search the upper half of the interval; otherwise, we search the lower half.

If job j is eligible for machine M_i and $p_j \leq s_i \cdot C$, then job j is *feasible* for machine M_i, and vice versa. In AssignB, U_i represents the set of unassigned jobs which are eligible for machine M_i and sorted in non-increasing order of their lengths, and L_i denotes the total length of the jobs assigned to machine M_i, $i = 1, 2, \ldots, m$. Informally speaking, AssignB tries to put as many as possible of the largest, not yet assigned feasible jobs on the deeper machines such that each machine completes no later than $2C$.

AssignB (C):

```
Step 1. Let $L_i = 0$, $U_i = J_i$, $i = 1, 2, \ldots, m$. Let $h$ be equal to the maximum depth of the machines in
        tree $RT$.
Step 2. While ($h > 0$), do:
    (i) For each machine $M_i$ whose depth is $h$, do:
    Find the first job $j \in U_i$ such that $p_j \leq s_i \cdot C$. Assign job $j$ and the jobs after it in $U_i$ to machine
    $M_i$ until $L_i > s_i \cdot C$ or the jobs after $j$ in $U_i$ are used up. Delete the newly assigned jobs from $U_i$.
```

Merge U_i into U_k, where M_k is the parent of M_i. (U_i and U_k are ordered sets before merging, and thus after merging U_k is still an ordered set.)

(ii) Let $h = h - 1$.

Step 3. If the total length of the jobs in U_m is no more than $2s_m \cdot C$, then process all the jobs in U_m on machine M_m (whose depth is zero), and the procedure succeeds and terminates. Otherwise, the procedure fails and terminates.

Lemma 2.4. *If $OPT \le C$, then AssignB will generate a feasible schedule for $Q|\mathcal{M}_j(tree)|C_{\max}$ with makespan at most $2C$ in $O(mn)$ time.*

Proof. Let Π^* and Π denote an optimal schedule and the schedule generated by AssignB, respectively. Since $OPT \le C$, in Π^*, machine M_i cannot process any job of length larger than $s_i \cdot C$. Hence, in Π, we let M_i process only the jobs whose lengths are no more than $s_i \cdot C$. We have $L_i \le 2s_i \cdot C$, and thus $C_i \le 2C$, $i = 1, 2, \dots, m$.

If $OPT \le C$, when AssignB terminates, it must be true that $U_m = \emptyset$. We prove this claim by contradiction. Suppose that some job j cannot be assigned when AssignB terminates. Let j be associated with the tree node M_{a_j}, such that its processing set $\mathcal{M}_j$ consists of the machines on the path from M_{a_j} to the root of the tree. Let $FM_j \subseteq \mathcal{M}_j$ denote the set of the feasible machines for job j, i.e., for any machine $M_l \in FM_j$, $p_j \le s_l \cdot C$. We have $L_l > s_l \cdot C$ for any machine $M_l \in FM_j$. Let D denote the set of the jobs processed on the machines in FM_j in Π. All the jobs in D have lengths at least p_j and hence cannot be assigned to the machines in $\mathcal{M}_j \backslash FM_j$. We consider the following two different cases:

Case 1. There are no eligible machines outside $\mathcal{M}_j$ for the jobs in D.

In this case, all the jobs in D have to be processed on the machines in FM_j in any optimal schedule. However, since $L_l > s_l \cdot C$ for any machine $M_l \in FM_j$, Π^* cannot complete all the jobs in D by time $C \ge OPT$, a contradiction.

Case 2. Some jobs in D have eligible machines in $\mathcal{M} \backslash \mathcal{M}_j$.

Some jobs in D may have eligible machines in $\mathcal{M} \backslash \mathcal{M}_j$, but they may be not feasible for these machines, or their feasible machines in $\mathcal{M} \backslash \mathcal{M}_j$ have not enough space left to accommodate them when they are assigned. Since AssignB assigns the largest feasible jobs on the machines in $\mathcal{M} \backslash \mathcal{M}_j$ greedily such that each machine completes later than C whenever possible, an optimal schedule cannot schedule the jobs any better than this on the machines in $\mathcal{M} \backslash \mathcal{M}_j$. Therefore, the total length of the jobs processed on the machines in FM_j in Π is a lower bound on the total length of the jobs processed on the machines in FM_j in Π^*. As in Case 1, since $L_l > s_l \cdot C$ for any machine $M_l \in FM_j$, Π^* cannot complete all the jobs in D by time $C \ge OPT$, a contradiction.

□

We use the binary search together with AssignB to solve $Q|\mathcal{M}_j(tree)|C_{\max}$. Then, we get the following:

Theorem 2.5. *There is a 2-approximation algorithm for $Q|\mathcal{M}_j(tree)|C_{\max}$ that runs in $O(mn \cdot \log(\sum_{j=1}^{n} p_j))$ time.*

3. 4/3-approximation algorithms

In this section we will get a 4/3-approximation algorithm for $Q|\mathcal{M}_j(inclusive)|C_{\max}$, and then extend it to solve $Q|\mathcal{M}_j(tree)|C_{\max}$. We continue to use the notations introduced in the previous section.

Let OPT denote the makespan of an optimal schedule for $Q|\mathcal{M}_j(inclusive)|C_{\max}$. We perform a binary search in $[LB, UB]$ to determine OPT, where $LB = \max_i AL_i$, $AL_i = \frac{\sum_{j\in\mathcal{J}_i\cup\mathcal{J}_{i+1}\cup\cdots\cup\mathcal{J}_m} p_j}{\sum_{l=i}^{m} s_l}$, and $UB = \frac{\sum_{j=1}^{n} p_j}{s_m}$. For each value C selected, we use the following procedure, AssignC1, to test whether it is possible to schedule the jobs such that the total length of the jobs processed on machine M_i is no more than $4s_i \cdot C/3$, $i = 1, 2, \ldots, m$.

If job j is eligible for machine M_i and $p_j \leq s_i \cdot C$, then job j is *feasible* for machine M_i, and vice versa.

For each value C selected, we classify feasible jobs as long, median, and short with respect to the machine speeds. For a particular machine M_i ($i = 1, 2, \ldots, m$), job j is *long* if $2s_i \cdot C/3 < p_j \leq s_i \cdot C$, or *median* if $s_i \cdot C/3 < p_j \leq 2s_i \cdot C/3$, or *short* if $p_j \leq s_i \cdot C/3$.

In AssignC1, U_i represents the set of unassigned jobs which are eligible for machine M_i and sorted in non-increasing order of their lengths, $i = 1, 2, \ldots, m$. In Step 2(ii) of AssignC1, we first compare the total length of the two largest median jobs in U_i with the length of the largest long job in U_i. If the former is larger, then we schedule the two largest median jobs in U_i on machine M_i. Otherwise, we schedule the largest long job in U_i on machine M_i.

AssignC1 (C):

Step 1. Let $U_0 = \varnothing$ and $i = 1$.

Step 2. While $i \leq m$, do:

(i) Merge U_{i-1} into J_i to get U_i.

(ii) If there is no long job in U_i, then add a "dummy" long job of length zero into U_i. Similarly, if there is no median job (or only one median job) in U_i, then add two (or one) "dummy" median job(s) of length zero into U_i. Let j_1, j_2 and j_3 denote the largest long job and the two largest median jobs in U_i, respectively. If $p_{j_2} + p_{j_3} > p_{j_1}$, then let j_2 and j_3 be processed on M_i and remove them from U_i; else, let j_1 be processed on M_i and remove it from U_i.

(iii) If the total length of the jobs on M_i is less than or equal to $s_i \cdot C$, then we repeatedly assign the largest unassigned short jobs in U_i to M_i until the first time that the total length of the jobs on M_i is larger than $s_i \cdot C$. Remove the newly assigned jobs from U_i.

(iv) Let $i = i + 1$.

Lemma 3.1. *If $OPT \leq C$, then AssignC1 will generate a feasible schedule for $Q|\mathcal{M}_j(inclusive)|C_{\max}$ with makespan at most $4C/3$ in $O(mn)$ time.*

Proof. Let Π^* be an optimal schedule. Let Π be the schedule generated by AssignC1. We will prove the lemma by modifying Π^* into Π.

To do so, we handle the machines in increasing order of their indices. Suppose that machines $M_1, M_2, \ldots, M_{i-1}$ have been handled. We illustrate how to handle machine M_i. At this point, the jobs on machines $M_1, M_2, \ldots, M_{i-1}$ in Π^* (after machines $M_1, M_2, \ldots, M_{i-1}$ have been handled) are processed in the same manner as they are processed on machines $M_1, M_2, \ldots, M_{i-1}$ in Π. We treat these jobs as assigned jobs. All the other jobs are treated as unassigned jobs. The unassigned jobs which are eligible for machine M_i form the set U_i.

As described in Step 2(ii) of AssignC1, if $p_{j_2} + p_{j_3} > p_{j_1}$, then we let j_2 and j_3 be processed on M_i in modified Π^*. To achieve this, if at least one part of j_2 is on machine M_l ($l > i$) other than M_i, we move this part from M_l to M_i, and move a corresponding part of the same length (consisting of parts of the jobs which are on M_i in Π^* but not on M_i in Π, cutting some job if necessary) from M_i to M_l. If the total length of the jobs which are on M_i in Π^* but not on M_i in Π is less than the length of this part of j_2, then we exchange this part of j_2 and all the jobs which are on M_i in Π^* but not on M_i in Π. Repeat this process until the entire j_2 appears on M_i in modified Π^*. Perform a similar exchange process until the entire j_3 appears on M_i in modified Π^*. Note that $p_{j_2} + p_{j_3} \leq 4s_i \cdot C/3$. If $p_{j_2} + p_{j_3} \leq p_{j_1}$, then we let j_1 be processed on M_i in modified Π^*, by performing a similar exchange process. Note that $p_{j_1} \leq s_i \cdot C$.

Next, we assign some short jobs in U_i on machine M_i as described in Step 2(iii) of AssignC1, by performing a similar exchange process. When we finish handling of M_i, the jobs processed on M_i in modified Π^* are those processed on M_i in Π. Moreover, the total length of the jobs on M_i is no more than $4s_i \cdot C/3$. Hence, M_i completes no later than $4C/3$.

Although some jobs have to be cut during the modification, when we finish handling of M_m, no cut job exists in modified Π^* (i.e., Π). It is easy to check that the total length of the jobs processed on machines $M_1, M_2, \ldots, M_i$ in Π is an upper bound on the total length of the jobs processed on machines $M_1, M_2, \ldots, M_i$ in unmodified Π^*, $i = 1, 2, \ldots, m$.

□

We can give an alternative implementation of AssignC1, called AssignC2, which runs in $O((m + n) \log n)$ time. The idea is to store the jobs in U_i in a balanced binary search tree T, using their lengths as the keys. Before we perform Step 2(ii) of AssignC1, we do the following. First, find the largest job of length no more than $s_i \cdot C$ in T. If the job does not exist, then let $i = i + 1$ and move on to the next iteration. If the job is a long job, then let it be j_1. Otherwise, let j_1 denote a "dummy" long job of length zero. Next, find the largest job of length no more than $2s_i \cdot C/3$ in T. If the job does not exist or is a short job, then let j_2 and j_3 denote two "dummy" median jobs of length zero. Otherwise (i.e., the job is a median job), let it be j_2, and continue to find another largest job of length no more than this job in T. If the job does not exist or is a short job, then let j_3 denote a "dummy" median job of length zero. Otherwise, let it be j_3. After j_1, j_2 and j_3 are determined, we perform Step 2(ii) of AssignC1. To perform Step 2(iii) of AssignC1, we need to find the currently largest job of length no more than $s_i \cdot C/3$ (i.e., the largest short job) in T. Note that each job will be inserted into T or deleted from T at most once. The balanced binary search tree T can be constructed and maintained in $O(n \log n)$ time. Moreover, determining j_1, j_2 and j_3 for each machine can be done in $O(\log n)$ time. Determining the currently largest short job in T can also be done in $O(\log n)$ time. Therefore, the overall running time of AssignC2 is $O((m + n) \log n)$.

Lemma 3.2. *If $OPT \leq C$, then AssignC2 will generate a feasible schedule for $Q|\mathcal{M}_j(inclusive)|C_{\max}$ with makespan at most $4C/3$ in $O((m + n) \log n)$ time.*

Theorem 3.3. *There is a 4/3-approximation algorithm for $Q|\mathcal{M}_j(inclusive)|C_{\max}$ that runs in $O(min\{mn, (m + n) \log n\} \cdot \log(\sum_{j=1}^{n} p_j))$ time.*

Next, we extend the algorithm to solve $Q|\mathcal{M}_j(tree)|C_{\max}$. Given the rooted tree RT whose nodes are the m machines, we index the nodes of RT in non-increasing order of their depths, ties broken in favor of the leftmost node. The root of RT is M_m.

Let OPT denote the optimal makespan for $Q|\mathcal{M}_j(tree)|C_{\max}$. For machine M_i (which represents a node of RT), let I_i denote the set of the indices of the machines on the path from M_i to the root of the tree. Let $AL_i = \frac{\sum_{l\in I_i}\sum_{j\in\mathcal{J}_l} p_j}{\sum_{l\in I_i} s_l}$ denote the average load on the machines whose indices are in I_i, $i = 1, 2, \ldots, m$. Let $LB = \max_i AL_i$. We have: $LB \le OPT$. Let $UB = \frac{\sum_{j=1}^{n} p_j}{s_m}$. We have: $OPT \le UB$.

We perform a binary search in $[LB, UB]$ to determine OPT. For each value C selected, we use the following procedure, AssignD, to test whether it is possible to schedule the jobs such that the total length of the jobs processed on machine M_i is no more than $4s_i \cdot C/3$, $i = 1, 2, \ldots, m$. We continue to use the related definitions for $Q|\mathcal{M}_j(inclusive)|C_{\max}$, such as feasible, long, median and short jobs.

In AssignD, U_i represents the set of unassigned jobs which are eligible for machine M_i and sorted in non-increasing order of their lengths, $i = 1, 2, \ldots, m$. In Step 2(i)(1) of AssignD, we first compare the total length of the two largest median jobs in U_i with the length of the largest long job in U_i. If the former is larger, then we schedule the two largest median jobs in U_i on machine M_i. Otherwise, we schedule the largest long job in U_i on machine M_i, $i = 1, 2, \ldots, m$.

AssignD (C):

Step 1. Let $U_i = J_i$, $i = 1, 2, \ldots, m$. Let h be equal to the maximum depth of the machines in tree RT.

Step 2. While ($h > 0$), do:

(i) For each machine M_i whose depth is h, do:

(1) If there is no long job in U_i, then add a "dummy" long job of length zero into U_i. Similarly, if there is no median job (or only one median job) in U_i, then add two (or one) "dummy" median job(s) of length zero into U_i. Let j_1, j_2 and j_3 denote the largest long job and the two largest median jobs in U_i, respectively. If $p_{j_2} + p_{j_3} > p_{j_1}$, then let j_2 and j_3 be processed on M_i and remove them from U_i; else, let j_1 be processed on M_i and remove it from U_i.

(2) If the total length of the jobs on M_i is less than or equal to $s_i \cdot C$, then we repeatedly assign the largest unassigned short jobs in U_i to M_i until the first time that the total length of the jobs on M_i is larger than $s_i \cdot C$. Remove the newly assigned jobs from U_i.

(3) Merge U_i into U_k, where M_k is the parent of M_i.

(ii) Let $h = h - 1$.

Step 3. If the total length of the jobs in U_m is no more than $4s_m \cdot C/3$, then process all the jobs in U_m on machine M_m, and the procedure succeeds and terminates. Otherwise, the procedure fails and terminates.

Similarly to the proof of Lemma 3.1, we can prove the following lemma.

Lemma 3.4. *If $OPT \le C$, then AssignD will generate a feasible schedule for $Q|\mathcal{M}_j(tree)|C_{\max}$ with makespan at most $4C/3$ in $O(mn)$ time.*

Theorem 3.5. *There is a 4/3-approximation algorithm for $Q|\mathcal{M}_j(tree)|C_{\max}$ that runs in $O(mn \cdot \log(\sum_{j=1}^{n} p_j))$ time.*

4. Conclusions

In this paper we investigated the problem of minimizing makespan on uniform machines with restricted assignment. We presented algorithms with approximation ratios 2 and 4/3 for the cases of inclusive and tree-hierarchical restrictions. Since the algorithms do not rely on the speed hierarchical assumption, they generalize the results presented in [7]. The running times of the algorithms contain a factor of $\log(\sum_{j=1}^{n} p_j)$, and in consequence they are not strongly polynomial time. To get strongly polynomial time algorithms, we use the technique described in [9] to modify the above algorithms slightly. The approximation ratios then become $2 + \varepsilon$ and $4/3 + \varepsilon$, where $\varepsilon > 0$ can be made arbitrarily small. In addition, as pointed out in [7], since the algorithms are based on the binary search, they produce schedules with makespans $2\lceil OPT \rceil$ or $4\lceil OPT \rceil/3$, where OPT denotes the optimal makespan.

It would be interesting to study the problem of minimizing makespan on uniform machines with other objective functions, or with other special types of assignment restrictions, such as nested, or interval restrictions. For further research, some learning strategies may be introduced, such as Probably Approximately Correct (PAC) learning with importance reweighting [22], dynamic feature weight selection [23], robust learning, granular-ball learning [24] or Complete Random Forest (CRF) based class noise filtering learning [25].

Acknowledgments

We thank the editor and reviewers for their helpful suggestions. This work is supported by the Natural Science Foundation of Shandong Province, China (No. ZR2020MA030).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. B. Chen, C. N. Potts, G. J. Woeginger, A review of machine scheduling: Complexity, algorithms and approximability, in *Handbook of combinatorial optimization*, Springer, (1998), 1493–1641. https://doi.org/10.1007/978-1-4613-0303-9_25
2. J. Y. Leung, *Handbook of scheduling: algorithms, models, and performance analysis*, CRC Press, 2004.
3. M. Drozdowski, Classic scheduling theory, in *Scheduling for Parallel Processing*, Springer, (2009), 55–86.
4. R. L. Graham, E. L. Lawler, J. K. Lenstra, A. R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Ann. Discrete Math.*, **5** (1979), 287–326. https://doi.org/10.1016/S0167-5060(08)70356-X
5. P. Brucker, Scheduling Algorithms, fifth edition, Springer, 2007.
6. J. Y. Leung, C. L. Li, Scheduling with processing set restrictions: A literature update, *Int.J. Prod. Econ.*, **175** (2016), 1–11. https://doi.org/10.1016/j.ijpe.2014.09.038

7. J. Y. Leung, Ng C, Fast approximation algorithms for uniform machine scheduling with processing set restrictions, *Eur. J. Oper. Res.*, **260** (2017), 507–513. https://doi.org/10.1016/j.ejor.2017.01.013

8. L. Epstein, A. Levin, Scheduling with processing set restrictions: PTAS results for several variants, *Int.J. Prod. Econ.*, **133** (2011), 586–595. https://doi.org/10.1016/j.ijpe.2011.04.024

9. J. Ou, J. Y. Leung, C. L. Li, Scheduling parallel machines with inclusive processing set restrictions, *Nav. Res. Log.*, **55** (2008), 328–338.

10. C. L. Li, X. Wang, Scheduling parallel machines with inclusive processing set restrictions and job release times, *Eur. J. Oper. Res.*, **200** (2010), 702–710. https://doi.org/10.1016/j.ejor.2009.02.011

11. D. G. Kafura, V. Y. Shen, Task scheduling on a multiprocessor system with independent memories, *SIAM J. Comput.*, **6** (1977), 167–187. https://doi.org/10.1137/0206014

12. H. C. Hwang, S. Y. Chang, K. Lee, Parallel machine scheduling under a grade of service provision, *Comput. Opera. Res.*, **31** (2004), 2055–2061. https://doi.org/10.1016/S0305-0548(03)00164-3

13. C. A. Glass, H. Kellerer, Parallel machine scheduling with job assignment restrictions, *Nav. Res. Log.*, **54** (2007), 250–257. https://doi.org/10.1002/nav.20202

14. A. Bar-Noy, A. Freund, J. Naor, On-line load balancing in a hierarchical server topology, *SIAM J. Comput.*, **31** (2001), 527–549. https://doi.org/10.1137/S0097539798346135

15. Y. Huo, J. T. Leung, Fast approximation algorithms for job scheduling with processing set restrictions, *Theor. Comput. Sci.*, **411** (2010), 3947–3955. https://doi.org/10.1016/j.tcs.2010.08.008

16. Y. Lin, W. Li, Parallel machine scheduling of machine-dependent jobs with unit-length, *Eur. J. Oper. Res.*, **156** (2004), 261–266. https://doi.org/10.1016/S0377-2217(02)00914-1

17. C. L. Li, Scheduling unit-length jobs with machine eligibility restrictions, *Eur. J. Oper. Res.*, **174** (2006), 1325–1328. https://doi.org/10.1016/j.ejor.2005.03.023

18. K. Lee, J. Y. Leung, M. L. Pinedo, Scheduling jobs with equal processing times subject to machine eligibility constraints, *J. Scheduling*, **14** (2011), 27–38. https://doi.org/10.1007/s10951-010-0190-0

19. C. L. Li, Q. Li, Scheduling jobs with release dates, equal processing times, and inclusive processing set restrictions, *J. Oper. Res. Soc.*, **66** (2015), 516–523. https://doi.org/10.1057/jors.2014.22

20. C. L. Li, K. Lee, A note on scheduling jobs with equal processing times and inclusive processing set restrictions, *J. Oper. Res. Soc.*, **67** (2016), 83–86. https://doi.org/10.1057/jors.2015.56

21. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, third edition, MIT press, 2009.

22. T. Liu, D. Tao, Classification with noisy labels by importance reweighting, *IEEE Trans. Pattern Anal. Mach. Intell.*, **38** (2015), 447–461. https://doi.org/10.1109/TPAMI.2015.2456899

23. Z. An, X. Wang, B. Li, Z. Xiang, B. Zhang, Robust visual tracking for UAVs with dynamic feature weight selection, *Appl. Intell.*, **2022** (2022), 1–14. https://doi.org/10.1007/s10489-022-03719-6

24. S. Xia, Y. Liu, X. Ding, G. Wang, H. Yu, Y. Luo, Granular ball computing classifiers for efficient, scalable and robust learning, *Infor. Sci.*, **483** (2019), 136–152. https://doi.org/10.1016/j.ins.2019.01.010

25. S. Xia, G. Wang, Z. Chen, Y. Duan, Complete random forest based class noise filtering learning for improving the generalizability of classifiers, *IEEE. Trans. Knowl. Data Eng.*, **31** (2018), 2063–2078. https://doi.org/10.1109/TKDE.2018.2873791