Theoretical Computer Science 507 (2013) 34–40

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Minimizing the total weighted completion time of fully parallel jobs with integer parallel units[☆]

Qiang Zhang[a], Weiwei Wu[b], Minming Li[a,*]

[a] Department of Computer Science, City University of Hong Kong, Hong Kong Special Administrative Region

[b] School of Computer Science and Engineering, Southeast University, China

ARTICLE INFO

Keywords:
Algorithms
Machine scheduling
Weighted completion time
Parallel jobs
Integer parallel units

ABSTRACT

We consider the total weighted completion time minimization in the following scheduling problem. There are m identical resources available at each time unit, and n jobs. Each job requires a number s_i of resources and one resource can only be assigned to one job at each time unit. Each job is also called fully parallel such that the job is satisfied once it receives enough resources no matter how the resources distribute. The objective is to find a schedule that minimizes $\sum w_i C_i$, where w_i is the weight of job J_i and C_i is the time when job J_i receives s_i resources. We show that the total weighted completion time minimization is NP-hard when m is an input of the problem. We then give a simple greedy algorithm with an approximation ratio 2. Finally, we present a polynomial time algorithm with complexity $O(n^{d+1})$ to solve this problem when the number of different resource requirements that are not multiples of m is at most d.

1. Introduction

Scheduling problems with the objective of minimizing total weighted completion time have been extensively studied in computer science and operations research. In the literature of computer science, a machine scheduling problem is denoted by a 3-tuple $\alpha|\beta|\gamma$ introduced in [10], where α denotes the machine environment, β denotes the additional constraints on the jobs, and γ denotes the objective function. The work in the machine scheduling originated from the classical problem $1||\sum w_i C_i$, where jobs are processed on a single machine and the objective is to minimize the total weighted completion time. Smith [19] showed that this problem could be solved optimally by a greedy algorithm, the Largest-Ratio-First rule particularly. When multiple machines are considered and each job is executed in a single machine, problem $Pm||\sum w_i C_i$ was shown to be NP-hard in [2] and a dynamic programming algorithm was given in [17]. Kawaguchi and Kyan [11] gave the worst case analysis of Largest-Ratio-First schedules. Later, Skutella and Woeginger [18] gave a PTAS for this problem and Conway et al. [4] solved the special case of this problem $Pm||\sum C_i$ when all jobs have the same weight. People also studied the total weighted completion time in the parallel scheduling model, namely $Pm|size_i|\sum w_i C_i$, where jobs must be executed on a number $size_i$ of machines at the same time. Fishkin et al. [7] gave a PTAS for this problem. Further with the introduction of release times, most of total weighted completion time minimization problems are NP-hard. Afrati et al. [1] gave PTASs for some classes of total weighted completion time minimization problems with release time.

In the literature of operations research, the total weighted completion time has been studied in different settings. For example, minimizing the total weighted completion time has been considered in the *concurrent open shop model*, denoted

☆ This work was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 124411].

* Corresponding author. Tel.: +852 34429538; fax: +852 34420503.
E-mail addresses: qianzhang8@student.cityu.edu.hk (Q. Zhang), weiweiwu@seu.edu.cn (W. Wu), minming.li@cityu.edu.hk (M. Li).

0304-3975/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.tcs.2013.02.017

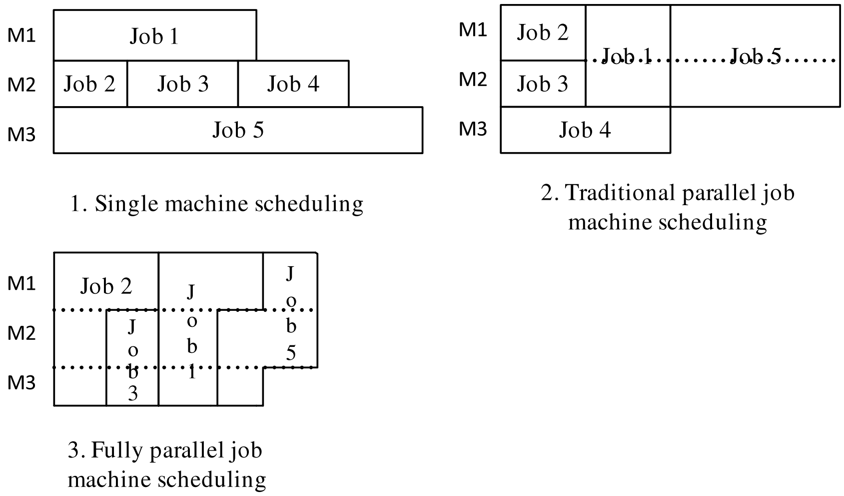

Fig. 1. Fundamental differences among three scheduling problems.

as $PD||\sum w_iC_i$, where there are a set of machines, with each machine responsible for one type of operation, and a set of jobs, with each job having a weight and requiring a specific number of processing for each type of operation. Several variants of this problem have been proved NP-hard [3,12,16,20]. Garg et al. [9] showed that $PD||\sum w_iC_i$ is APX-hard and recently Mastrolilli [13] gave a primal–dual 2-approximation algorithm for this problem.

Resource allocations are also studied by economists. One of the most important problems in economics is social welfare maximization. In markets, agents have different valuations when different bundles of items are assigned. The social welfare maximization problem aims to find out an assignment of items that maximizes the total valuation of all agents. Scheduling problems can be considered as different variants of social welfare maximization problem when each job has values for different bundles of processing units. The social welfare maximization problem is in general NP-hard and it is well-known that computing an optimal solution for social welfare maximization requires an exponential number of queries even in the general queries model [15]. Therefore, most works on this topic focus on the approximability of this problem. It is known that there is a $(1-1/e)$-approximation for the general submodular welfare maximization problem in a stronger demand oracle model [5], which was improved to $(1-1/e+\epsilon)$ in [6]. Mirrokni et al. [14] gave the tight lower bounds for the welfare maximization via value query model in combinatorial auctions when the valuation function is submodular, subadditive and superadditive.

In this paper, we consider the total weighted completion time in the following scheduling model. There are a set of identical resources available at each time unit and a set of jobs. Each job needs a number of resources and has the cost based on the completion time (i.e. the time it gets all resources). The fundamental problem in this model is how to allocate these resources to the jobs such that the social cost is minimized. This model is motivated by real-life examples. Suppose that a steel factory has a fixed amount of steel production each week and ships out the steel at the end of each week. The factory receives orders from a lot of customers. Each of them needs a specific amount of steel. The question left to the factory is how to allocate its steel to customers in order to minimize the social cost under the supply constraint.

We model the scheduling problem in the context of machine scheduling by introducing the concept of *fully parallel jobs*. In the traditional parallel scheduling model $Pm|size_i|\sum w_iC_i$, the parallel jobs must be executed on a number $size_i$ of machines at the same time. In other words, each parallel job is assigned to a time × machine rectangle (see Fig. 1). In contrast, we study the fully parallel jobs which are allowed to be scheduled arbitrarily at any number of machines at different time slots. The job is satisfied once it has been processed for a number of time slots it requires regardless it is continuously or cumulatively scheduled. The aim of our scheduling problem is to minimize the total weighted completion time of fully parallel jobs. We denote the problem as $Pm|arbitrary|\sum w_iC_i$ by 3-tuple notation. Fig. 1 illustrates the fundamental differences among the single job ($Pm||\sum w_iC_i$), parallel job ($Pm|size_i|\sum w_iC_i$) and fully parallel job machine scheduling ($Pm|arbitary|\sum w_iC_i$) problems.

This is the first time that the machine scheduling problem of fully parallel jobs has been studied. Similar to the other machine scheduling problems, we use the total weighted completion time as the objective in this work. We assume that the time unit is equal to the job processing unit and a single time unit can only be assigned to one job. Otherwise, problem $Pm|arbitrary|\sum w_iC_i$ can be solved optimally by the Largest-Ratio-First rule used in problem $1||\sum w_iC_i$. The main contributions of this work are the analysis of complexity of this machine scheduling problem and the design of exact and approximation algorithms to solve it. We first prove that it is NP-hard to obtain the optimal schedule when the number of machines available at each time unit is the input of problem. The proof is based on a reduction from the 3-partition problem. We then show a greedy algorithm that computes a 2-approximation schedule. Finally, we design a $O(n^{d+1})$ polynomial time algorithm to compute the optimal solution when the number of different job sizes that are not multiples of the number of machines is at most d.

The rest of the paper is organized as follows. The basic notations and definitions are described in Section 2. The NP-hardness result on the scheduling problem is presented in Section 3. Several characterizations of the optimal solution are given in Section 4. In Section 5, we show that the Largest-Ratio-First greedy algorithm gives a 2-approximation solution. In Section 6, we give a dynamic programming algorithm to solve the problem optimally when the number of different job sizes, which are not multiples of the number of machines, is bound by a constant. Finally, we conclude our work by the discussions on the results and possible future work in Section 7.

2. Definitions and notations

Let m denote the number of identical machines available at every single time $t > 0$. There are n fully parallel jobs which are indexed as $J_1, J_2, \ldots, J_n$. Job J_i requires $s_i \in \mathbb{Z}^+$ processing units and has weight $w_i \in \mathbb{R}^+$. We also say s_i is the size of job J_i. Without loss of generality, we assume that $\frac{w_1}{s_1} \geq \frac{w_2}{s_2} \geq \cdots \geq \frac{w_n}{s_n}$. Otherwise, it takes $O(n \log n)$ time to sort the jobs in non-increasing order of $\frac{w}{s}$. Let $J(k)$ denote the set of jobs whose sizes are k where k is not a multiple of m, that is, $J(k) = \{J_i | s_i = k, k \text{ is not a multiple of } m\}$, and $J(m)$ denotes the set of jobs whose sizes are the multiples of m, that is, $J(m) = \{J_i | s_i = zm, \text{ for some integer } z\}$. Let $d = |\{k | J(k) \neq \emptyset \text{ and } k \text{ is not a multiple of } m\}|$ be the number of different sizes that are not multiples of m.

A schedule is a tuple $(t_{11}, t_{12}, \ldots, t_{1m}, \ldots)$ where the processing unit on machine y at time x is assigned to job $t_{xy} \in \{1, 2, \ldots n\}$. We say a job is satisfied when it receives the required number of processing units. A schedule is feasible if all jobs are satisfied, that is, $\forall i \in \{1, 2, \ldots n\}, |\{(x, y) | t_{xy} = i\}| = s_i$. For a certain schedule A, let r_i^A be the time that J_i receives its first processing unit and C_i^A be the time that J_i gets all the required number of processing units. We say job J_j starts at time r_i^A and finishes at time C_i^A. We also say that job J_i is scheduled before J_j if $r_i^A < r_j^A$.

Schedule A incurs cost $w_i C_i^A$ for J_i. The objective of the $Pm|arbitrary|\sum w_i C_i$ problem is to find a feasible schedule that minimizes total weighted completion time $\sum w_i C_i$. Let OPT be the optimal solution. Hence, the minimum total weighted completion time is $\sum w_i C_i^{OPT}$. We say schedule A is β-approximation if $\sum w_i C_i^A \leq \beta \sum w_i C_i^{OPT}$. When the context is clear, the symbol A would be omitted on the above notations.

3. Analysis of complexity

In this section, we show that the problem $Pm|arbitrary|\sum w_i C_i$ is NP-hard by a reduction from the well-known 3-partition problem.

Definition 3.1 (*3-Partition Problem*). Given a positive integer b and a set $S = \{a_1, a_2, \ldots, a_n\}$ of $n = 3k$ positive integers such that $b/4 < a_i < b/2$ and $\sum_{j=1}^{n} a_j = kb$, the problem is to determine whether S can be partitioned into k subsets $S_1, S_2, \ldots, S_k$ such that each subset contains 3 elements and the sum of the numbers in each subset is equal to b.

Garey and Johnson [8] proved the 3-partition problem to be NP-complete.

Definition 3.2 (*Fully Scheduled*). For a certain schedule A, we say machines at time t are *fully scheduled* if the total size of jobs that start and finish at time t equals the number of machines, i.e. $\sum_{J_j | r_j^A = t \text{ and } C_j^A = t} s_j = m$.

Lemma 3.1. *Assume that there are m machines available at each time, n jobs with $w_i = s_i$ and $\sum s_i = mt$ where $t > 0$, then the total weighted completion time is at least $\frac{1+t}{2} mt$ in any feasible schedule.*

Proof. Let v_i denote the total size of jobs finished at time i in any schedule. Since there are only m machines available at each time, we have $\sum_{i=1}^{j} v_i \leq jm, 0 \leq j \leq t$. The total weighted completion time is $\sum_{i=1}^{t} i v_i$. We also know $\sum_{i=1}^{t} v_i = \sum s_i = mt$ because all the jobs could finish at time t. The minimum total weighted completion time occurs when all machines are *fully scheduled* at every time unit, i.e. $v_i = m$ for all i. Therefore, the total weighted completion time is at least $\sum_{i=1}^{t} im = \frac{1+t}{2} mt$. □

Theorem 3.1. *Problem $Pm|arbitrary|\sum w_i C_i$ is NP-hard when the number of machines is also the input of the problem.*

Proof. We prove this theorem by a reduction from the 3-partition problem to problem $Pm|arbitrary|\sum w_i C_i$. For an arbitrary instance $\mathcal{I}_1$ in 3-partition problem, we create an instance $\mathcal{I}_2$ of $Pm|arbitrary|\sum w_i C_i$ in the following way. For any positive number $a_i \in S$ in $\mathcal{I}_1$, we create a corresponding job J_i such that $w_i = s_i = a_i$ in $\mathcal{I}_2$. There are b machines available at every time unit in $\mathcal{I}_2$. It is clear that this construction takes polynomial time.

First, we show that any partition solution in $\mathcal{I}_1$ can be mapped to an optimal schedule in $\mathcal{I}_2$. Suppose that a partition $S_1, S_2, \ldots, S_k$ is the solution of instance $\mathcal{I}_1$, then the corresponding schedule A in $\mathcal{I}_2$ assigns the jobs in set $\{J_i | s_i \in S_t\}$ to all the b machines at time t. Since all the jobs finishing on time k have total weight b, the total weighted completion time $\sum w_i C_i^A$ is $bk\frac{k+1}{2}$. Since $w_i = s_i$ in $\mathcal{I}_2$, by Lemma 1, we know that any feasible schedule A' in $\mathcal{I}_2$ would have total weighted completion time at least $bk\frac{k+1}{2}$. Therefore, schedule A is the optimal solution in $\mathcal{I}_2$.

Second, we show that the optimal schedule in $\mathcal{I}_2$ suggests the answer to $\mathcal{I}_1$. Suppose that A is the optimal schedule in $\mathcal{I}_2$, since $w_i = s_i$ in $\mathcal{I}_2$ and $\sum s_i = kb$, any feasible schedule A' in $\mathcal{I}_2$ would have a total weighted completion time of at least $bk\frac{k+1}{2}$. The case $\sum w_i C_i^{A'} = bk\frac{k+1}{2}$ occurs if and only if the machines are *fully scheduled* at every time unit. Since $b/4 < s_i < b/4$, there are exactly 3 jobs at every time unit if machines are fully scheduled. Therefore, the answer to 3-partition problem $\mathcal{I}_1$ is to check whether the total weighted completion time $\sum w_i C_i^A$ equals $bk\frac{k+1}{2}$. If it is the case, the partition solution for $\mathcal{I}_1$ can be constructed by assigning the numbers in set $\{s_i | C_i^A = t\}$ to S_t in $\mathcal{I}_1$. □

4. Characterizations of the optimal schedule

In this section, we give some fundamental observations on the structure of the optimal solution to problem $Pm|arbitrary|\sum w_i C_i$, which will lead us to design and analyze the algorithms in the following sections.

Lemma 4.1. *There exists an optimal schedule where a job receives processing units if all jobs before it finish on or before this time unit, i.e., $C_i \le r_j$ or $C_j \le r_i$ for any jobs J_i and J_j.*

Proof. We give a constructive proof for this lemma. Suppose A is a schedule where J_j receives processing unit before J_i finishes and J_i receives processing unit before J_j finishes, that is, $C_i > r_j$ and $C_j > r_i$. Another schedule A' can be constructed as follows. W.l.o.g, assume $C_j \ge C_i$. First, keep processing units assignment unchanged except J_i and J_j in A. Then assign the first s_i processing units assigned to job J_i and J_j to job J_i and remaining s_j processing units to J_j. Note that in A' job J_i will not finish later than in A, that is, $C_i^{A'} \le C_i^A$, and J_j finishes at the same time as in A. Therefore, $\sum w_i C_i^{A'} \le \sum w_i C_i^A$. Then the lemma directly follows. □

Lemma 4.1 allows us to only concentrate on the schedules in which all jobs receive the processing units in a non-preemptive way when we design and analyze algorithms for problem $Pm|arbitrary|\sum w_i C_i$ since the optimal cost can be computed once the processing order of the jobs are fixed. From now on, we can simply describe a schedule in terms of the processing order of the jobs, i.e. schedule A is an n-tuple $(t_1, \ldots, t_n)$ that specifies the processing order of the jobs where job t_i is the i-th job scheduled in schedule A.

Lemma 4.2. *In the optimal solution, for any two jobs $J_i, J_j \in J(m)$, if $\frac{w_i}{s_i} > \frac{w_j}{s_j}$, then $C_i^{OPT} \le C_j^{OPT}$.*

Proof. We prove this lemma by contradiction. Suppose that OPT is the optimal schedule which minimizes $\sum w_i C_i$ and $C_i^{OPT} > C_j^{OPT}$. By Lemma 4.1, it is sufficient to only consider the case where $C_j^{OPT} \le r_i^{OPT}$. Let B be the set of jobs scheduled between C_j^{OPT} and r_i^{OPT}. If B is empty, it is easy to see that OPT can be improved by swapping job J_i and J_j because of $\frac{w_i}{s_i} > \frac{w_j}{s_j}$. This contradicts the optimality of OPT. Now let us consider the case that B is not empty. Since OPT is the optimal solution, we know that advancing B before J_j will not reduce the total weighted completion time. Since s_j is a multiple of m, it implies that $\frac{w_j}{s_j/m} \ge \frac{\sum_{b\in B} w_b}{t_1}$ where s_j/m is the time that jobs in the set B advance and t_1 is the time that J_j delays if advancing B before J_j. By the similar argument, advancing J_i before B will not reduce the total weighted completion time, we get $\frac{w_i}{s_i/m} \le \frac{\sum_{b\in B} w_b}{t_2}$ where s_i/m is the time that jobs in the set B delay and t_2 is the time that J_i advances if advancing J_i before B. Since both s_i and s_j are multiples of m, it is easy to verify that $t_1 = t_2$. Therefore, we have $\frac{w_j}{s_j/m} \ge \frac{\sum_{b\in B} w_b}{t_2} \ge \frac{w_i}{s_i/m}$ which contradicts the assumption $\frac{w_i}{s_i} > \frac{w_j}{s_j}$. □

Lemma 4.3. *In the optimal solution, for any two jobs J_i, J_j and k, if $J_i, J_j \in J(k)$ and $w_i > w_j$ ($\frac{w_i}{s_i} > \frac{w_j}{s_j}$ equivalently since $s_i = s_j$), then $C_i^{OPT} \le C_j^{OPT}$.*

Proof. We prove this lemma by contradiction. Suppose OPT is the optimal schedule to minimize $\sum_i w_i C_i$ and job J_i finishes later than J_j in OPT, i.e. $C_i^{OPT} > C_j^{OPT}$. The total weighted completion time can be reduced $(w_i - w_j)(C_i^{OPT} - C_j^{OPT})$ by swapping the processing unit assignments for J_i and J_j in schedule OPT. It contradicts the fact that OPT is the optimal schedule. □

Lemmas 4.2 and 4.3 show that the job with larger ratio of $\frac{w}{s}$ is scheduled first in the optimal solution when jobs have the equal size or their sizes are the multiples of the number of machines. We will use Lemmas 4.2 and 4.3 in Section 6 to design an efficient algorithm to solve $Pm|arbitrary|\sum w_i C_i$ when d is a constant.

5. Approximation algorithm

The classical Largest-Ratio-First (LRF) rule is one of the well-known scheduling algorithms and has been examined in different scheduling problems. LRF schedule is always favored in the practice due to its simplicity. In this section we show that the LRF rule gives a 2-approximation schedule for problem $Pm|arbitrary|\sum w_i C_i$.

Definition 5.1 (*Largest-Ratio-First Rule*)**.** Largest-Ratio-First (LRF) rule outputs a schedule where time units are assigned to the jobs in non-increasing order of $\frac{w}{s}$.

Theorem 5.1. *LRF rule gives a 2-approximation schedule for problem $Pm|arbitrary|\sum w_i C_i$, i.e. $\sum w_i C_i^{LRF} \le 2\sum w_i C_i^{OPT}$.*

We prove Theorem 5.1 by examining the lower bound of the optimal schedule and the upper bound of the LRF schedule in the follow lemmas.

Lemma 5.1. *$\sum w_i C_i^{OPT}$ is at least $\frac{1}{2}\sum w_i(\frac{\sum_{j\le i} s_j}{m}+1)$.*

Proof. Let A^* denote the optimal schedule in problem $Pm|arbitrary|\sum w_iC_i$. We first prove a lower bound that the minimum total weighted completion time $\sum_i w_iC_i^{A^*}$ is at least the optimal cost in the following minimization problem $\mathcal{P}$. The input consists of the same n jobs as those in our problem and m machines are available at every time unit. Each machine can only be assigned to one job at a time. Instead of minimizing $\sum w_iC_i$, the objective is to find a schedule of jobs S that minimizes $\sum w_i \frac{\sum_{J_j\in F_S(i)} s_j}{m}$, where $F_S(i)$ is the set of jobs scheduled before J_i including J_i itself in schedule S.

It is easy to verify that minimization problem $\mathcal{P}$ is equivalent to problem $1||\sum w_iC_i$ which is the weighted completion time minimization problem in single machine scheduling. Therefore, LRF schedule gives the optimal objective value $\sum w_i \frac{\sum_{j\le i} s_j}{m}$ for problem $\mathcal{P}$. Since $\mathcal{P}$ and $Pm|arbitrary|\sum w_iC_i$ problem share the same set of feasible schedules and the C_i^S in $Pm|arbitrary|\sum w_iC_i$ is greater than or equal to the $\frac{\sum_{J_j\in F_S(i)} s_j}{m}$ in $\mathcal{P}$ for every feasible schedule S, the minimum total weighted completion time $\sum w_iC_i^{A^*}$ is lower bounded by $\sum w_j \frac{\sum_{i\le j} s_i}{m}$.

Another lower bound $\sum_i w_i$ directly follows since C_i is at least 1 for any job in any feasible schedule. Combining the two lower bounds completes the proof. □

Lemma 5.2. *$\sum w_iC_i^{LRF}$ is at most $\sum w_i\left(\frac{\sum_{j\le i} s_j}{m}+\frac{m-1}{m}\right)$.*

Proof. Since the jobs are scheduled by their orders in the LRF schedule and there are m machines available at every time unit, we have the following

$$\sum w_iC_i^{LRF} = \sum w_i\left\lceil \frac{\sum_{j\le i} s_j}{m}\right\rceil.$$

The lemma follows by bounding $\lceil\frac{\sum_{j\le i} s_i}{m}\rceil$ by $\frac{\sum_{j\le i} s_j}{m}+\frac{m-1}{m}$. □

The following example shows that the approximation ratio for the LRF schedule is tight. Suppose that there are two jobs and m machines available at each time unit. Let $w_1 = 1+\epsilon, s_1 = 1$ and $w_2 = m, s_2 = m$. The total weighted completion time from LRF schedule is $(1+\epsilon)+2m$ and the minimum total weighted completion time is $m+2(1+\epsilon)$. Therefore, the LRF schedule is $(2-\epsilon^{'})$ approximation for arbitrarily small ϵ'.

6. Limited sizes

We showed that $Pm|arbitrary|\sum w_iC_i$ is NP-hard in general and the LRF rule would give a 2-approximation schedule. In this section we present a polynomial time algorithm with complexity $O(n^{d+1})$ to find the optimal solution when d is a constant. Recall that d is the number of different job sizes that are not multiples of m. This scenario happens when the scheduler has the prior information on the job sizes. For example, the jobs sizes are limited to certain numbers in the application. We say that the jobs in the same set $J(k)$ form a group. Therefore, there are at most $d+1$ groups. We assume that the jobs in the same group are listed in non-increasing order of $\frac{w}{s}$. Let g_i^j be the j-th job in group i and $|g_i|$ be the size of group i.

Now let us recall the characterizations established in Section 4. Lemma 4.1 says that a schedule can be computed once the processing order of jobs is known. Lemmas 4.2 and 4.3 strength the result by suggesting that the optimal schedule can be even computed once the order of groups the scheduled jobs come from is known. By easily associating the k-th scheduled job to one of the $d+1$ groups, there is an exponential algorithm with complexity $O((d+1)^n)$ to compute the optimal solution. For constant value d, the merit of this section is to show that the optimal solution can be computed with a significantly lower complexity $O(n^{d+1})$.

The technique of our algorithm for this restricted problem comes from the dynamic programming. The intuition is to construct the optimal schedule by recursively applying Lemmas 4.2 and 4.3 with the non-preemptive property of the optimal solution. As mentioned, a scheduled can be computed once the order of groups the schedule jobs belong to is known. Our algorithm examines all possible orders of groups by adding one more job recursively and favors the schedule with smaller total cost. We present our dynamic programming in Algorithm 1. Let $c(a_1,\ldots,a_{d+1})$ denote the minimum total weighted completion time where a_i jobs have been scheduled in group i and the number of scheduled jobs in each group cannot exceed the number of jobs in this group, i.e. for all i, $a_i \le |g_i|$. The dynamic programming is initialized at $c(0,\ldots,0)=0$. Recall that a schedule is determined once the execution order of groups is determined by Lemma 4.1, the schedule corresponding to the minimum total weighted completion time can be easily constructed. Moreover, by the recursive relationship, it is easy to see that $c(a_1,\ldots,a_{d+1}) = \min_i c(a_1,\ldots,a_i-1,\ldots,a_{d+1}) + w_{g_i^{a_i}}C_{g_i^{a_i}}$ where $C_{g_i^{a_i}}$ is the completion time of the a_i^{th} job in group i by appending it after the previous schedule. By recursively computing $c(\cdot)$, the minimum total weighted completion time directly follows.

Algorithm 1:

Input: A set of jobs $J_1, J_2, \ldots, J_n$ is divided into $d+1$ groups based on their sizes
Output: The minimum total weighted completion time $\sum w_i C_i$

Initialization:
// initialize the base case
$c(0, \ldots, 0) = 0$;
$k_i = |g_i|, \forall i$;

Main:
// compute the minimum total weighted completion time by procedure c.
$\min \sum w_i C_i = c(k_1, \ldots, k_{d+1})$;

Procedure $c(a_1, \ldots, a_{d+1})$**:**
// H contains the indexes of the groups in $c(\cdot)$ which are not zero
$H = \{i | a_i \geq 1\}$;
// append job $g_i^{a_i}$ into the previous schedule $c(a_1, \ldots, a_i - 1, \ldots, a_{d+1})$ and compute the resulting total weighted completion time.
// return the one with minimum total weighted completion time
return $\min_{i \in H} c(a_1, \ldots, a_i - 1, \ldots, a_{d+1}) + w_{g_i^{a_i}} C_{g_i^{a_i}}$;

Lemma 6.1. *Algorithm* 1 *computes a schedule that minimizes the cost* $\sum w_i C_i$.

Proof. Recall that Lemmas 4.2 and 4.3 are the necessary conditions for any optimal schedule that minimizes total weighted completion time $\sum w_i C_i$. Algorithm 1 examines all the schedules satisfying Lemmas 4.2 and 4.3 and outputs the schedule with smallest total weighted completion time. Therefore, it gives an optimal schedule. □

Lemma 6.2. *Algorithm* 1 *terminates in* $O(n^{d+1})$ *time.*

Proof. We analyze the running time of our algorithm by counting the size of dynamic programming in Algorithm 1. For a particular number $l \leq n$, there are at most $\binom{l+d}{d}$ schedules such at $\sum a_i = l$. Each schedule needs $d+1$ computations to get the minimum. Therefore, there are at most $(d+1)\binom{l+d}{d}$ computations. Since l can only take n different values, the total required computations are bounded by $O(n^{d+1})$. It completes the proof. □

7. Conclusion and discussion

In this paper we introduce the fully parallel jobs and consider the total weighted completion time as the objective in the machine scheduling with fully parallel jobs. We prove that total weighted completion time minimization is NP-hard in general and show that the LRF rule is 2-approximation. We also give a polynomial time algorithm to compute the optimal solution when the sizes of the jobs are restricted. It would be interesting to explore other classes of this problem where algorithms can be designed to compute the optimal solution. It is also interesting to design the polynomial time approximation scheme (PTAS) for the general case, or prove that it is APX-hard. Furthermore, with the introduction of release time of each job in the online environment, we believe that the problem could be more challenging.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, M. Sviridenkom, Approximation schemes for minimizing average weighted completion time with release dates, in: Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, 1999, pp. 32–43.
[2] J. Bruno, E. Coffman Jr., R. Sethi, Scheduling independent tasks to reduce mean finishing time, Communications of the ACM 17 (7) (1974) 382–387.
[3] Z. Chen, N. Hall, Supply chain scheduling: assembly systems, Tech. rep., Department of Systems Engineering, University of Pennsylvania, 2001.
[4] R. Conway, W. Maxwell, L. Miller, Theory of Scheduling, Dover Publications, 2003.
[5] S. Dobzinski, M. Schapira, An improved approximation algorithm for combinatorial auctions with submodular bidders, in: Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithm, 2006, pp. 1064–1073.
[6] U. Feige, J. Vondrak, Approximation algorithms for allocation problems: improving the factor of 1–1/e, in: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 667–676.
[7] A. Fishkin, K. Jansen, L. Porkolab, On minimizing average weighted completion time: a ptas for scheduling general multiprocessor tasks, in: Proceedings of the 13th International Symposium on Fundamentals of Computation Theory, 2001, pp. 495–507.
[8] M. Garey, D. Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM Journal on Computing 4 (1975) 397–411.
[9] N. Garg, A. Kumar, V. Pandit, Order scheduling models: hardness and algorithms, in: Foundations of Software Technology and Theoretical Computer Science, 2007, FSTTCS 2007, pp. 96–107.
[10] R. Graham, E. Lawler, J. Lenstra, A. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of discrete Mathematics 5 (2) (1979) 287–326.
[11] T. Kawaguchi, S. Kyan, Worst case bound of an lrf schedule for the mean weighted flow-time problem, SIAM Journal on Computing 15 (4) (1986) 1119–1129.

[12] J. Leung, H. Li, M. Pinedo, Order scheduling in an environment with dedicated resources in parallel, Journal of Scheduling 8 (5) (2005) 355–386.
[13] M. Mastrolilli, M. Queyranne, A. Schulz, O. Svensson, N. Uhan, Minimizing the sum of weighted completion times in a concurrent open shop, Operations Research Letters 38 (5) (2010) 390–395.
[14] V. Mirrokni, M. Schapira, J. Vondrák, Tight information-theoretic lower bounds for welfare maximization in combinatorial auctions, in: Proceedings of the 9th ACM Conference on Electronic Commerce, 2008, pp. 70–77.
[15] N. Nisan, I. Segal, The communication requirements of efficient allocations and supporting prices, Journal of Economic Theory 129 (1) (2006) 192–224.
[16] T. Roemer, A note on the complexity of the concurrent open shop problem, Journal of scheduling 9 (4) (2006) 389–396.
[17] S. Sahni, Algorithms for scheduling independent tasks, Journal of the ACM (JACM) 23 (1) (1976) 116–127.
[18] M. Skutella, G. Woeginger, A ptas for minimizing the weighted sum of job completion times on parallel machines, in: Proceedings of the 31th Annual ACM Symposium on Theory of Computing, ACM, 1999, pp. 400–407.
[19] W. Smith, Various optimizers for single-stage production, Naval Research Logistics Quarterly 3 (1–2) (1956) 59–66.
[20] C. Sung, S. Yoon, Minimizing total weighted completion time at a pre-assembly stage composed of two feeding machines, International Journal of Production Economics 54 (3) (1998) 247–255.